

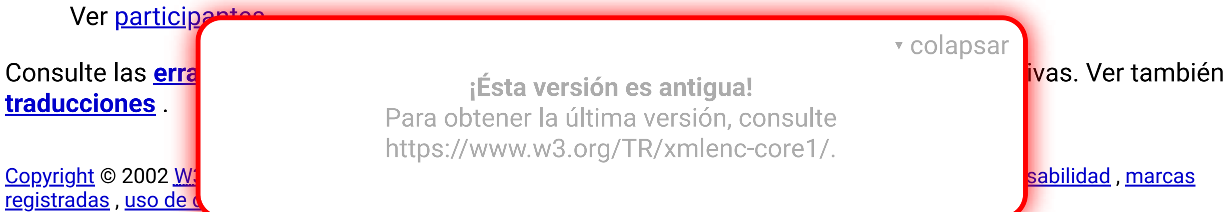
Sintaxis y procesamiento de cifrado XML

Recomendación del W3C del 10 de diciembre de 2002

Esta versión:
http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/
Ultima versión:
http://www.w3.org/TR/xmlenc-core/
Versión previa:
http://www.w3.org/TR/2002/PR-xmlenc-core-20021003/

Editores
Donald Eastlake <dee3@torque.pothole.com>
José Reagle <reagle@w3.org>
Autores
Takeshi Imamura <IMAMU@jp.ibm.com>
Blair Dillaway <blaird@microsoft.com>
Ed Simón <edsimon@xmlsec.com>
Colaboradores
Ver participa

Consulte las **erra** ivas. Ver también **traducciones** .



Abstracto

Este documento especifica un proceso para cifrar datos y representar el resultado en XML. Los datos pueden ser datos arbitrarios (incluido un documento XML), un elemento XML o contenido de un elemento XML. El resultado del cifrado de datos es un elemento de cifrado XML que contiene o hace referencia a los datos cifrados.

Estado de este documento

Este documento es la Recomendación de cifrado XML (REC) del W3C . Este documento ha sido revisado por miembros del W3C y otras partes interesadas y ha sido respaldado por el Director como Recomendación del W3C. Es un documento estable y puede usarse como material de referencia o citarse como referencia normativa de otro documento. El papel del W3C al elaborar la Recomendación es llamar la atención sobre la especificación y promover su implementación generalizada. Esto mejora la funcionalidad y la interoperabilidad de la Web.

Esta especificación fue producida por el Grupo de Trabajo de Cifrado XML del W3C (Actividad), que cree que la especificación es suficiente para la creación de implementaciones interoperables independientes como se demuestra en el Informe de Interoperabilidad.

Las divulgaciones de patentes relevantes para esta especificación se pueden encontrar en la página de divulgación de patentes del Grupo de Trabajo de conformidad con la política del W3C.

Informe los errores en este documento a xml-encryption@w3.org (archivo público).

La lista de errores conocidos en esta especificación está disponible en http://www.w3.org/Encryption/2002/12-xmlenc-errata .

La versión en inglés de esta especificación es la única versión normativa. La información sobre las traducciones de este documento (si corresponde) está disponible en http://www.w3.org/Encryption/2002/12-xmlenc-translations .

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/.

Table of Contents

---

1 Introduction

This document specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption `EncryptedData` element which contains (via one of its children's content) or identifies (via a URI reference) the cipher data.

When encrypting an XML element or element content the `EncryptedData` element replaces the element or content (respectively) in the encrypted version of the XML document.

When encrypting arbitrary data (including entire XML documents), the `EncryptedData` element may become the root of a new XML document or become a child element in an application-chosen XML document.

1.1 Editorial and Conformance Conventions

This specification uses XML schemas [XML-schema] to describe the content model.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in RFC2119 [KEYWORDS]:

> "they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)"

Consequently, we use these capitalized keywords to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. These key words are not used (capitalized) to describe XML grammar; schema definitions unambiguously describe such requirements and we wish to reserve the prominence of these terms for the natural language descriptions of protocols and features. For instance, an XML attribute might be described as being "optional." Compliance with the XML-namespace specification [XML-NS] is described as "REQUIRED."

1.2 Design Philosophy

The design philosophy and requirements of this specification (including the limitations related to instance validity) are addressed in the XML Encryption Requirements [EncReq].

1.3 Versions, Namespaces, URIs, and Identifiers

No provision is made for an explicit version number in this syntax. If a future version is needed, it will use a different namespace. The experimental XML namespace [XML-NS] URI that MUST be used by implementations of this (dated) specification is:

```
xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
```

This namespace is also used as the prefix for algorithm identifiers used by this specification. While applications MUST support XML and XML namespaces, the use of internal entities [XML, section 4.2.1], the "`xenc`" XML namespace prefix [XML-NS, section 2] and defaulting/scoping conventions are OPTIONAL; we use these facilities to provide compact and readable examples. Additionally, the entity `&xenc;` is defined so as to provide short-hand identifiers for URIs defined in this specification. For example "`&xenc;Element`" corresponds to "http://www.w3.org/2001/04/xmlenc#Element".

This specification makes use of the XML Signature [XML-DSIG] namespace and schema definitions

```
xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
```

Los URI [URI] DEBEN cumplir con la definición de tipo [XML-Schema] y la especificación [XML-DSIG , 4.3.3.1 El atributo URI] (es decir, caracteres permitidos, escape de caracteres, soporte de esquema, etc.).`anyURI`

1.4 Agradecimientos

Se agradecen las contribuciones de los siguientes miembros del Grupo de Trabajo a esta especificación de acuerdo con las políticas de contribuyentes y la lista activa del Grupo de Trabajo .

- joseph ashwood
- Simon Blake-Wilson, Certicom
- Frank D. Cavallito, Sistemas BEA
- Eric Cohen, PricewaterhouseCoopers
- Blair Dillaway, Microsoft (Autor)
- Blake Dournaee, Seguridad RSA
- Donald Eastlake, Motorola (Editor)
- Barb Fox, Microsoft
- Christian Geuer-Pollmann, Universidad de Siegen
- Tom Gindin, IBM
- Jiandong Guo, Faos

- Phillip Hallam-Baker, Verisign
- Amir Herzberg, NewGenPay
- Merlín Hughes, Baltimore
- Federico Hirsch
- Maryann Hondo, IBM
- Takeshi Imamura, IBM (Autor)
- Mike Just, Entrust, Inc.
- Brian La Macchia, Microsoft
- Hiroshi Maruyama, IBM
- John Messing, Ley en línea
- Shivaram Mysore, Sun Microsystems
- Thane Plambeck, Verisign
- Joseph Reagle, W3C (Presidente, Editor)
- Alexei Sanin
- Jim Schaad, consultoría Soaring Hawk
- Ed Simon, XMLsec (Autor)
- Daniel Toth, Ford
- Yongge Wang, Certicom
- Steve Wiley, mi prueba

Además, agradecemos a las siguientes personas por sus comentarios durante y después de la última llamada:

- Martín Durst, W3C
- Dan Lanz, Zolera
- Susan Lesch, W3C
- David Orchard, Sistemas BEA
- Ronald Rivest, MIT

2 Descripción general y ejemplos de cifrado (no normativo)

Esta sección proporciona una descripción general y ejemplos de la sintaxis de cifrado XML. La sintaxis formal se encuentra en *Sintaxis de cifrado* (sección 3); el procesamiento específico se proporciona en *las Reglas de procesamiento* (sección 4).

Expresado en forma abreviada, el `EncryptedData`elemento tiene la siguiente estructura (donde "?" denota cero o una ocurrencia; "+" denota una o más ocurrencias; "*" denota cero o más ocurrencias; y la etiqueta de elemento vacía significa que el elemento debe ser vacío):

```
<¿Identificación de datos cifrados? ¿Tipo? ¿Tipo de Mimica? ¿Codificación?>
  <Método de cifrado/>?
  <ds:Información clave>
    <Clave cifrada>?
    <Método de acuerdo>?
    <ds:NombreClave>?
    <ds:Método de recuperación>?
    <ds:*>?
  </ds:KeyInfo>?
  <Datos cifrados>
    <ValorCifrado>?
    <¿URI de referencia de cifrado?>?
  </CipherData>
  <Propiedades de cifrado>?
</EncryptedData>
```

El `CipherData`elemento envuelve o hace referencia a los datos cifrados sin procesar. Si es envolvente, los datos cifrados sin procesar son el `CipherValue`contenido del elemento; si se hace referencia, el atributo `CipherReference`del elemento `URI`apunta a la ubicación de los datos cifrados sin procesar

2.1 Granularidad del cifrado

Consider the following fictitious payment information, which includes identification information and information appropriate to a payment method (e.g., credit card, money transfer, or electronic check):

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

This markup represents that John Smith is using his credit card with a limit of $5,000USD.

2.1.1 Encrypting an XML Element

Smith's credit card number is sensitive information! If the application wishes to keep that information confidential, it can encrypt the `CreditCard` element:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
   xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <CipherData>
      <CipherValue>A23B45C56</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

By encrypting the entire `CreditCard` element from its start to end tags, the identity of the element itself is hidden. (An eavesdropper doesn't know whether he used a credit card or money transfer.) The `CipherData` element contains the encrypted serialization of the `CreditCard` element.

2.1.2 Encrypting XML Element Content (Elements)

As an alternative scenario, it may be useful for intermediate agents to know that John used a credit card with a particular limit, but not the card's number, issuer, and expiration date. In this case, the content (character data or children elements) of the `CreditCard` element is encrypted:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
     Type='http://www.w3.org/2001/04/xmlenc#Content'>
      <CipherData>
        <CipherValue>A23B45C56</CipherValue>
      </CipherData>
    </EncryptedData>
  </CreditCard>
</PaymentInfo>
```

2.1.3 Encrypting XML Element Content (Character Data)

Or, consider the scenario in which all the information *except* the actual credit card number can be in the clear, including the fact that the Number element exists:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>
      <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
       Type='http://www.w3.org/2001/04/xmlenc#Content'>
        <CipherData>
          <CipherValue>A23B45C56</CipherValue>
        </CipherData>
      </EncryptedData>
    </Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Both `CreditCard` and `Number` are in the clear, but the character data content of `Number` is encrypted.

2.1.4 Cifrado de datos arbitrarios y documentos XML

Si el escenario de la aplicación requiere que toda la información esté cifrada, todo el documento se cifra como una secuencia de octetos. Esto se aplica a datos arbitrarios, incluidos documentos XML.

```
<?xml versión='1.0'?>
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
 MimeType='texto/xml'>
  <Datos cifrados>
```

```
      <CipherValue>A23B45C56</CipherValue>
    </a> CipherDat_
  </a> EncryptedDat_
```

2.1.5 Supercifrado : cifrado de datos cifrados

Un documento XML puede contener cero o más `EncryptedData` elementos. `EncryptedData`no puede ser padre o hijo de otro `EncryptedData`elemento. Sin embargo, los datos reales cifrados pueden ser cualquier cosa, incluidos `EncryptedData`elementos `EncryptedKey` (es decir, supercifrado). Durante el supercifrado de un elemento `EncryptedData`o `EncryptedKey`, se debe cifrar todo el elemento. Cifrar solo el contenido de estos elementos o cifrar elementos secundarios seleccionados es una instancia no válida según el esquema proporcionado.
Por ejemplo, considere lo siguiente:

```
  <p ay:PaymentInfoxmlns:pay='http://example.org/pagov2'>
    <Id. de datos cifrados='ED1' xmlns='http://www.w3.org/2001/04/xmlenc#'
     Tipo =' http://www.w3.org/2001/04/xmlenc#Element '>
      <Datos cifrados>
        <CipherValue> originalDatos cifrados</CipherValue>
      </CipherData>
    </EncryptedData>
  </pago:InfoPago>
```

Un supercifrado válido de " `//xenc:EncryptedData[@Id='ED1']`" sería:

```
  <p ay:PaymentInfoxmlns:pay='http://example.org/pagov2'>
    <Id. de datos cifrados='ED2' xmlns='http://www.w3.org/2001/04/xmlenc#'
     Tipo =' http://www.w3.org/2001/04/xmlenc#Element '>
      <Datos cifrados>
        <CipherValue> newDatos cifrados</CipherValue>
      </a> CipherDat_
    </a> EncryptedDat_
  </o> pay:PaymentInf_
```

donde el `CipherValue`contenido de ' `newEncryptedData`' es la codificación base64 de la secuencia de octetos cifrados resultante del cifrado del `EncryptedData`elemento con `Id='ED1'`.

2.2 `EncryptedData`y `EncryptedKey` uso

2.2.1 `EncryptedData`con clave simétrica (`KeyName`)

```
  [s1] <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
        Tipo =' http://www.w3.org/2001/04/xmlenc#Element '/>
  [s2] <Método de cifrado
          Algoritmo='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
  [s3] <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
  [s4] <ds:KeyName>John Smith</ ds:KeyName>
  [s5] </ds:KeyInfo>
  [s6] <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
  [s7] </EncryptedData>
```

`[s1]`El tipo de datos cifrados se puede representar como un valor de atributo para ayudar en el descifrado y el procesamiento posterior. En este caso, los datos cifrados eran un "elemento". Otras alternativas incluyen el "contenido" de un elemento o una secuencia de octetos externa que también puede identificarse mediante los atributos `MimeType`y `Encoding`.

`[s2]`Este (3DES CBC) es un cifrado de clave simétrica.

`[s4]`La clave simétrica tiene un nombre asociado "John Smith".

`[s6]` `CipherData`contiene un `CipherValue`, que es una secuencia de octetos codificada en base64. Alternativamente, podría contener un `CipherReference`, que es una referencia de URI junto con las transformaciones necesarias para obtener los datos cifrados como una secuencia de octetos.

2.2.2 `EncryptedKey` (`ReferenceList`, `ds:RetrievalMethod`, `CarriedKeyName`)

La siguiente `EncryptedData`estructura es muy similar a la anterior, excepto que esta vez se hace referencia a la clave mediante `ds:RetrievalMethod`:

```
  [t01] <Id. de datos cifrados='ED'
         xmlns='http://www.w3.org/2001/04/xmlenc#'>
  [t02] <Método de cifrado
           Algoritmo='http://www.w3.org/2001/04/xmlenc#aes128-cbc'/>
```

```
[t03] <d s:KeyInfoxmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
[t04] <ds: RetrievalMethodURI='#EK'
         Escriba="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
[t05] <ds:KeyName>Sally Doe</ds:KeyName>
[t06] </ds:KeyInfo>
[t07] <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
[t08] </EncryptedData>
```

`[t02]`Este (AES-128-CBC) es un cifrado de clave simétrica.

`[t04]` `ds:RetrievalMethod`se utiliza para indicar la ubicación de una clave con tipo `&xenc;EncryptedKey`. La clave (AES) se encuentra en '#EK'.

`[t05]` `ds:KeyName`proporciona un método alternativo para identificar la clave necesaria para descifrar el archivo `CipherData`. Cualquiera o ambos `ds:KeyName`y `ds:KeyRetrievalMethod` podrían usarse para identificar la misma clave.

Dentro del mismo documento XML, existía una `EncryptedKey` estructura a la que se hacía referencia en `[t04]`:

```
[t09] <Id. de clave cifrada='EK' xmlns='http://www.w3.org/2001/04/xmlenc#'>
[t10] <Método de cifrado
       Algoritmo="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
[t11] <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
[t12] <ds:KeyName>John Smith</ ds:KeyName>
[t13] </ds:KeyInfo>
[t14] <CipherData><CipherValue>xyzabc</CipherValue></CipherData>
[t15] <Lista de referencias>
[t16] <URI de referencia de datos='#ED'/>
[t17] </ListaReferencia>
[t18] <CarriedKeyName>Sally Doe</CarriedKeyName>
[t19] </EncryptedKey>
```

`[t09]`El `EncryptedKey`elemento es similar al `EncryptedData`elemento excepto que los datos cifrados son siempre un valor clave.

`[t10]`Es `EncryptionMethod`el algoritmo de clave pública RSA.

`[t12]` `ds:KeyName` of "John Smith" is a property of the key necessary for decrypting (using RSA) the `CipherData`.

`[t14]` The `CipherData`'s `CipherValue` is an octet sequence that is processed (serialized, encrypted, and encoded) by a referring encrypted object's `EncryptionMethod`. (Note, an EncryptedKey's `EncryptionMethod` is the algorithm used to encrypt these octets and does not speak about what type of octets they are.)

`[t15-17]` A `ReferenceList` identifies the encrypted objects (`DataReference` and `KeyReference`) encrypted with this key. The `ReferenceList` contains a list of references to data encrypted by the symmetric key carried within this structure.

`[t18]` The `CarriedKeyName` element is used to identify the encrypted key value which may be referenced by the `KeyName` element in `ds:KeyInfo`. (Since ID attribute values must be unique to a document,`CarriedKeyName` can indicate that several `EncryptedKey` structures contain the same key value encrypted for different recipients.)

3 Encryption Syntax

This section provides a detailed description of the syntax and features for XML Encryption. Features described in this section MUST be implemented unless otherwise noted. The syntax is defined via [XML-Schema] with the following XML preamble, declaration, internal entity, and import:

```
Schema Definition:

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE schema  PUBLIC "-//W3C//DTD XMLSchema 200102//EN"
 "http://www.w3.org/2001/XMLSchema.dtd"
 [
   <!ATTLIST schema
     xmlns:xenc CDATA #FIXED 'http://www.w3.org/2001/04/xmlenc#'
     xmlns:ds CDATA #FIXED 'http://www.w3.org/2000/09/xmldsig#'>
   <!ENTITY xenc 'http://www.w3.org/2001/04/xmlenc#'>
   <!ENTITY % p ''>
   <!ENTITY % s ''>
  ]>

<schema xmlns='http://www.w3.org/2001/XMLSchema' version='1.0'
        xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
        xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
        targetNamespace='http://www.w3.org/2001/04/xmlenc#'
        elementFormDefault='qualified'>
```

```
  <import namespace='http://www.w3.org/2000/09/xmldsig#'
          schemaLocation='http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd'/>
```

3.1 The `EncryptedType` Element

`EncryptedType` is the abstract type from which `EncryptedData` and `EncryptedKey` are derived. While these two latter element types are very similar with respect to their content models, a syntactical distinction is useful to processing. Implementation MUST generate laxly schema valid [XML-schema] `EncryptedData` or `EncryptedKey` as specified by the subsequent schema declarations. (Note the laxly schema valid generation means that the content permitted by `xsd:ANY` need not be valid.) Implementations SHOULD create these XML structures (`EncryptedType` elements and their descendents/content) in Normalization Form C [NFC, NFC-Corrigendum].

```
Schema Definition:

<complexType name='EncryptedType' abstract='true'>
  <sequence>
    <element name='EncryptionMethod' type='xenc:EncryptionMethodType'
             minOccurs='0'/>
    <element ref='ds:KeyInfo' minOccurs='0'/>
    <element ref='xenc:CipherData'/>
    <element ref='xenc:EncryptionProperties' minOccurs='0'/>
  </sequence>
  <attribute name='Id' type='ID' use='optional'/>
  <attribute name='Type' type='anyURI' use='optional'/>
  <attribute name='MimeType' type='string' use='optional'/>
  <attribute name='Encoding' type='anyURI' use='optional'/>
 </complexType>
```

`EncryptionMethod` is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail.

`ds:KeyInfo` is an optional element, defined by [XML-DSIG], that carries information about the key used to encrypt the data. Subsequent sections of this specification define new elements that may appear as children of `ds:KeyInfo`.

`CipherData` is a mandatory element that contains the `CipherValue` or `CipherReference` with the encrypted data.

`EncryptionProperties` can contain additional information concerning the generation of the `EncryptedType` (e.g., date/time stamp).

`Id` is an optional attribute providing for the standard method of assigning a string id to the element within the document context.

`Type` is an optional attribute identifying type information about the plaintext form of the encrypted content. While optional, this specification takes advantage of it for mandatory processing described in *Processing Rules: Decryption* (section 4.2). If the `EncryptedData` element contains data of `Type` 'element' or element 'content', and replaces that data in an XML document context, it is strongly recommended the `Type` attribute be provided. Without this information, the decryptor will be unable to automatically restore the XML document to its original cleartext form.

`MimeType` is an optional (advisory) attribute which describes the media type of the data which has been encrypted. The value of this attribute is a string with values defined by [MIME]. For example, if the data that is encrypted is a base64 encoded PNG, the transfer `Encoding` may be specified as 'http://www.w3.org/2000/09/xmldsig#base64' and the `MimeType` as 'image/png'. This attribute is purely advisory; no validation of the `MimeType` information is required and it does not indicate the encryption application must do any additional processing. Note, this information may not be necessary if it is already bound to the identifier in the `Type` attribute. For example, the Element and Content types defined in this specification are always UTF-8 encoded text.

3.2 The EncryptionMethod Element

EncryptionMethod is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail.

```
Schema Definition:

<complexType name='EncryptionMethodType' mixed='true'>
  <sequence>
    <element name='KeySize' minOccurs='0' type='xenc:KeySizeType'/>
    <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
    <any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
```

```
    <attribute name='Algorithm' type='anyURI' use='required'/>
  </complexType>
```

The permitted child elements of the `EncryptionMethod` are determined by the specific value of the `Algorithm` attribute URI, and the `KeySize` child element is always permitted. For example, the [RSA-OAEP algorithm](#) (section 5.4.2) uses the `ds:DigestMethod` and `OAEPparams` elements. (We rely upon the `ANY` schema construct because it is not possible to specify element content based on the value of an attribute.)

La presencia de cualquier elemento secundario `EncryptionMethod` que no esté permitido por el algoritmo o la presencia de un `KeySize`elemento secundario inconsistente con el algoritmo DEBE tratarse como un error. (Todos los URI de algoritmo especificados en este documento implican un tamaño de clave, pero esto no es cierto en general. Los algoritmos de cifrado de flujo más populares utilizan claves de tamaño variable).

3.3 El `CipherData`elemento

Es `CipherData`un elemento obligatorio que proporciona los datos cifrados. Debe contener la secuencia de octetos cifrados como texto codificado en base64 del `CipherValue`elemento o proporcionar una referencia a una ubicación externa que contenga la secuencia de octetos cifrados a través del `CipherReference`elemento.

```
  Definición del esquema:

  <elemento nombre=' CipherData' tipo=' xenc:CipherDataType'/>
  <nombre de tipo complejo=' CipherDataType'>
     <elección>
       <elemento nombre=' CipherValue' tipo='base64Binary'/>
       <elemento ref=' xenc:CipherReference'/>
     </elección>
   </tipocomplejo>
```

3.3.1 El `CipherReference`elemento

Si `CipherValue`no se suministra directamente, `CipherReference`identifica una fuente que, cuando se procesa, produce la secuencia de octetos cifrada.

El valor real se obtiene de la siguiente manera. Contiene `CipherReference URI`un identificador al que se le ha desreferenciado. Si el `CipherReference`elemento contiene una secuencia OPCIONAL de `Transforms`, los datos resultantes de desreferenciar el URI se transforman según lo especificado para producir el valor de cifrado deseado. Por ejemplo, si el valor está codificado en base64 dentro de un documento XML; las transformaciones podrían especificar una expresión XPath seguida de una decodificación base64 para extraer los octetos.

La sintaxis de `URI`y `Transforms`es similar a la de [[XML-DSIG](#)]. Sin embargo, existe una diferencia entre el procesamiento de firma y cifrado. En [[XML-DSIG](#)], tanto el procesamiento de generación como el de validación comienzan con los mismos datos de origen y realizan esa transformación en el mismo orden. En el cifrado, el descifrador sólo tiene los datos cifrados y las transformaciones especificadas se enumeran para el descifrador, en el orden necesario para obtener los octetos. En consecuencia, debido a que tiene una semántica diferente, `Transforms`está en el `&xenc;`espacio de nombres.

Por ejemplo, si el valor de cifrado relevante se captura dentro de un `CipherValue`elemento dentro de un documento XML diferente, `CipherReference`podría tener el siguiente aspecto:

```
  <CipherReference URI="http://www.example.com/CipherValues.xml">
    <Transformaciones>
      <ds:Transformar
       Algoritmo="http://www.w3.org/TR/1999/REC-xpath-19991116">
          <ds:XPath xmlns:rep="http://www.example.org/repositorio">
            self::text()[padre::rep:CipherValue[@Id="ejemplo1"]]
          </ds:XPath>
      </ds:Transformar>
      <ds:Algoritmo de transformación="http://www.w3.org/2000/09/xmldsig#base64"/>
    </Transforma>
  </CipherReference>
```

Las implementaciones DEBEN admitir la `CipherReference`función y la misma codificación URI, desreferenciación, esquema y códigos de respuesta HTTP que los de [[XML-DSIG](#)]. La `Transform`característica y los algoritmos de transformación particulares son OPCIONALES.

```
  Definición del esquema:

  <elemento nombre=' CipherReference' tipo=' xenc:CipherReferenceType'/>
   <nombre de tipo complejo=' CipherReferenceType'>
       <secuencia>
         <elemento nombre='Transforms' tipo='xenc:TransformsType' minOccurs='0'/>
       </secuencia>
```

```
      <nombre de atributo='URI' tipo='cualquierURI' uso='requerido'/>
  </tipocomplejo>

   <nombre de tipo complejo='Tipo de transformación'>
      <secuencia>
        <elemento ref='ds:Transform' maxOccurs='ilimitado'/>
      </secuencia>
    </tipocomplejo>
```

3.4 El `EncryptedData`elemento

El `EncryptedData`elemento es el elemento central de la sintaxis. Su `CipherData`elemento secundario no solo contiene los datos cifrados, sino que también es el elemento que reemplaza al elemento cifrado o sirve como raíz del nuevo documento.

```
Definición del esquema:

<elemento nombre=' EncryptedData' tipo=' xenc:EncryptedDataType'/>
<nombre de tipo complejo=' EncryptedDataType'>
  <Contenido complejo>
   <base de extensión=' xenc:EncryptedType'>
   </extensión>
  </complexContent>
</tipocomplejo>
```

3.5 Extensiones al`ds:KeyInfo` elemento

Hay tres formas de `CipherData`proporcionar el material de claves necesario para descifrar:

1. El elemento `EncryptedData`o `EncryptedKey`especifica el material de clave asociado a través de un elemento secundario de `ds:KeyInfo`. Todos los elementos secundarios de ds: `KeyInfo`especificados en [XML-DSIG] PUEDEN usarse como calificados:
   1. La compatibilidad con `ds:KeyValue`es OPCIONAL y puede usarse para transportar claves públicas, como los valores clave Diffie-Hellman (sección 5.5.1). (Obviamente NO SE RECOMIENDA incluir la clave de descifrado de texto plano, ya sea una clave privada o secreta).
   2. Se RECOMIENDA el soporte de `ds:KeyName`para hacer referencia a un .`EncryptedKey` `CarriedKeyName`
   3. `ds:RetrievalMethod`Se REQUIERE soporte para el mismo documento .

   Además, proporcionamos dos elementos secundarios adicionales: las aplicaciones DEBEN ser compatibles `EncryptedKey` (sección 3.5.1) y PUEDEN ser compatibles `AgreementMethod`(sección 5.5).
2. Un elemento separado (no dentro de `ds:KeyInfo`) `EncryptedKey`puede especificar el `EncryptedData`o `EncryptedKey`al cual se aplicará su clave descifrada a través de `DataReference`o `KeyReference`(sección 3.6).
3. El material de claves lo puede determinar el destinatario según el contexto de la aplicación y, por lo tanto, no es necesario mencionarlo explícitamente en el XML transmitido.

3.5.1 El `EncryptedKey`elemento

Identificador

`Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"`

(This can be used within a `ds:RetrievalMethod` element to identify the referent's type.)

The `EncryptedKey` element is used to transport encryption keys from the originator to a known recipient(s). It may be used as a stand-alone XML document, be placed within an application document, or appear inside an `EncryptedData` element as a child of a `ds:KeyInfo` element. The key value is always encrypted to the recipient(s). When `EncryptedKey` is decrypted the resulting octets are made available to the `EncryptionMethod` algorithm without any additional processing.

```
Schema Definition:

<element name='EncryptedKey' type='xenc:EncryptedKeyType'/>
<complexType name='EncryptedKeyType'>
  <complexContent>
    <extension base='xenc:EncryptedType'>
      <sequence>
        <element ref='xenc:ReferenceList' minOccurs='0'/>
        <element name='CarriedKeyName' type='string' minOccurs='0'/>
      </sequence>
      <attribute name='Recipient' type='string' use='optional'/>
    </extension>
  </complexContent>
</complexType>
```

ReferenceList is an optional element containing pointers to data and keys encrypted using this key. The reference list may contain multiple references to EncryptedKey and EncryptedData elements. This is done using KeyReference and DataReference elements respectively. These are defined below.

CarriedKeyName is an optional element for associating a user readable name with the key value. This may then be used to reference the key using the ds:KeyName element within ds:KeyInfo. The same CarriedKeyName label, unlike an ID type, may occur multiple times within a single document. The value of the key is to be the same in all EncryptedKey elements identified with the same CarriedKeyName label within a single XML document. Note that because whitespace is significant in the value of the ds:KeyName element, whitespace is also significant in the value of the CarriedKeyName element.

Recipient is an optional attribute that contains a hint as to which recipient this encrypted key value is intended for. Its contents are application dependent.

The Type attribute inheritted from EncryptedType can be used to further specify the type of the encrypted key if the EncryptionMethod Algorithm does not define a unambiguous encoding/representation. (Note, all the algorithms in this specification have an unambigous representation for their associated key structures.)

3.5.2 The ds:RetrievalMethod Element

The ds:RetrievalMethod [XML-DSIG]with a Type of 'http://www.w3.org/2001/04/xmlenc#EncryptedKey' provides a way to express a link to an EncryptedKey element containing the key needed to decrypt the CipherData associated with an EncryptedData or EncryptedKey element. The ds:RetrievalMethod with this type is always a child of the ds:KeyInfo element and may appear multiple times. If there is more than one instance of a ds:RetrievalMethod in a ds:KeyInfo of this type, then the EncryptedKey objects referred to must contain the same key value, possibly encrypted in different ways or for different recipients.

```
Schema Definition:

<!--
    <attribute name='Type' type='anyURI' use='optional'
    fixed='http://www.w3.org/2001/04/xmlenc#EncryptedKey' />
-->
```

3.6 The ReferenceList Element

ReferenceList is an element that contains pointers from a key value of an EncryptedKey to items encrypted by that key value (EncryptedData or EncryptedKey elements).

```
Schema Definition:

<element name='ReferenceList'>
  <complexType>
    <choice minOccurs='1' maxOccurs='unbounded'>
      <element name='DataReference' type='xenc:ReferenceType'/>
      <element name='KeyReference' type='xenc:ReferenceType'/>
    </choice>
  </complexType>
</element>

<complexType name='ReferenceType'>
  <sequence>
    <any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
  <attribute name='URI' type='anyURI' use='required'/>
</complexType>
```

DataReference elements are used to refer to EncryptedData elements that were encrypted using the key defined in the enclosing EncryptedKey element. Multiple DataReference elements can occur if multiple EncryptedData elements exist that are encrypted by the same key.

KeyReference elements are used to refer to EncryptedKey elements that were encrypted using the key defined in the enclosing EncryptedKey element. Multiple KeyReference elements can occur if multiple EncryptedKey elements exist that are encrypted by the same key.

For both types of references one may optionally specify child elements to aid the recipient in retrieving the EncryptedKey and/or EncryptedData elements. These could include information such as XPath transforms, decompression transforms, or information on how to retrieve the elements from a document storage facility. For example:

```
<ReferenceList>
  <DataReference URI="#invoice34">
```

```
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
          <ds:XPath xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
              self::xenc:EncryptedData[@Id="example1"]
          </ds:XPath>
        </ds:Transform>
      </ds:Transforms>
    </DataReference>
  </ReferenceList>
```

3.7 The `EncryptionProperties` Element

Identifier

`Type="http://www.w3.org/2001/04/xmlenc#EncryptionProperties"`

(This can be used within a `ds:Reference` element to identify the referent's type.)

Se pueden colocar elementos de información adicional relacionados con la generación de `EncryptedData`o en un elemento (por ejemplo, marca de fecha/hora o el número de serie del hardware criptográfico utilizado durante el cifrado). El atributo identifica la estructura que se describe. permite la inclusión de atributos del espacio de nombres XML que se incluirán (es decir, , y).`EncryptedKeyEncryptionPropertyTargetEncryptedTypeanyAttributexml:spacexml:langxml:base`

```
  Definición del esquema:

  <elemento nombre='EncryptionProperties' tipo='xenc:EncryptionPropertiesType'/>
  <complexType nombre='EncryptionPropertiesType'>
    <secuencia>
      <elemento ref='xenc:EncryptionProperty' maxOccurs='ilimitado'/>
    </secuencia>
    <nombre del atributo='Id' tipo='ID' uso='opcional'/>
  </tipocomplejo>

    <elemento nombre='EncryptionProperty' tipo='xenc:EncryptionPropertyType'/>
    <complexType nombre='EncryptionPropertyType' mixto='verdadero'>
      <elección maxOccurs='ilimitado'>
        <cualquier espacio de nombres='##other' ProcessContents='lax'/>
      </elección>
      <nombre del atributo='Destino' tipo='cualquierURI' uso='opcional'/>
      <nombre del atributo='Id' tipo='ID' uso='opcional'/>
      <anyAttribute namespace="http://www.w3.org/XML/1998/namespace"/>
    </tipocomplejo>
```

4 reglas de procesamiento

Esta sección describe las operaciones que se realizarán como parte del procesamiento de cifrado y descifrado mediante implementaciones de esta especificación. Los requisitos de conformidad se especifican en los siguientes roles:

Solicitud
: La aplicación que solicita la implementación de un cifrado XML mediante el suministro de datos y parámetros necesarios para su procesamiento.

cifrador
: Una implementación de cifrado XML con la función de cifrar datos.

Descifrador
: Una implementación de cifrado XML con la función de descifrar datos.

4.1 Cifrado

Para que cada elemento de datos se cifre como `EncryptedData`o `EncryptedKey`(elementos derivados de `EncryptedType`), el **cifrador** debe:

1. Seleccione el algoritmo (y los parámetros) que se utilizarán para cifrar estos datos.
2. Obtener y (opcionalmente) representar la clave.
    1. Si la clave debe identificarse (mediante nombre, URI o incluirse en un elemento secundario), construya la clave `ds:KeyInfo`según corresponda (p. ej. `ds:KeyName`, `ds:KeyValue`, `ds:RetrievalMethod`, etc.)
    2. Si se va a cifrar la clave en sí, construya un `EncryptedKey`elemento aplicando recursivamente este proceso de cifrado. El resultado puede ser hijo de `ds:KeyInfo`o puede existir en otro lugar y puede identificarse en el paso anterior.
3. cifrar los datos
    1. Si los datos son un ' elemento ' [XML , sección 3] o un elemento ' contenido ' [XML , sección 3.1], obtenga los octetos serializando los datos en UTF-8 como se especifica en [XML]. (La aplicación

DEBE proporcionar datos XML en [NFC].) La serialización PUEDE ser realizada por el **cifrador** . Si el **cifrador** no se serializa, entonces la **aplicación** DEBE realizar la serialización.
2. Si los datos son de cualquier otro tipo que aún no sean octetos, la **aplicación** DEBE serializarlos como octetos.
3. Cifre los octetos utilizando el algoritmo y la clave de los pasos 1 y 2.
4. A menos que el **descifrador** conozca implícitamente el tipo de datos cifrados, el **cifrado** DEBE proporcionar el tipo para la representación.

   La definición de este tipo como vinculado a un identificador especifica cómo obtener e interpretar los octetos de texto plano después del descifrado. Por ejemplo, el identificador podría indicar que los datos son una instancia de otra aplicación (por ejemplo, alguna aplicación de compresión XML) que debe procesarse más. O, si los datos son una secuencia de octetos simple, PUEDEN describirse con los atributos `MimeType`y `Encoding`. Por ejemplo, los datos pueden ser un documento XML (`MimeType="text/xml"`), una secuencia de caracteres (`MimeType="text/plain"`) o datos de imagen binaria (`MimeType="image/png"`).

4. Construya la estructura `EncryptedType`(`EncryptedData`o `EncryptedKey`):

   Una `EncryptedType`estructura representa toda la información analizada anteriormente, incluido el tipo de datos cifrados, el algoritmo de cifrado, los parámetros, la clave, el tipo de datos cifrados, etc.

   1. Si la secuencia de octetos cifrada obtenida en el paso 3 se va a almacenar en el `CipherData`elemento dentro de `EncryptedType`, entonces la secuencia de octetos cifrada se codifica en base64 y se inserta como contenido de un `CipherValue`elemento.
   2. Si la secuencia de octetos cifrada se va a almacenar externamente a la `EncryptedType`estructura, almacene o devuelva la secuencia de octetos cifrada y represente el URI y las transformaciones (si las hay) necesarias para que el descifrador recupere la secuencia de octetos cifrada dentro de un `CipherReference`elemento.
5. Procesar datos cifrados
   1. If the `Type` of the encrypted data is 'element' or element 'content', then the **encryptor** MUST be able to return the `EncryptedData` element to the **application**. The **application** MAY use this as the top-level element in a new XML document or insert it into another XML document, which may require a re-encoding.

      The **encryptor** SHOULD be able to replace the unencrypted 'element' or 'content' with the EncryptedData element. When an **application** requires an XML element or content to be replaced, it supplies the XML document context in addition to identifying the element or content to be replaced. The **encryptor** removes the identified element or content and inserts the `EncryptedData` element in its place.

      (Note: If the `Type` is "content" the document resulting from decryption will not be well-formed if (a) the original plaintext was not well-formed (e.g., PCDATA by itself is not well-formed) and (b) the `EncryptedData` element was previously the root element of the document)

   2. If the `Type` of the encrypted data *is not* 'element' or element 'content', then the **encryptor** MUST always return the `EncryptedData` element to the **application**. The **application** MAY use this as the top-level element in a new XML document or insert it into another XML document, which may require a re-encoding.

4.2 Decryption

For each `EncryptedType` derived element, (i.e., `EncryptedData` or `EncryptedKey`), to be decrypted, the **decryptor** must:

1. Process the element to determine the algorithm, parameters and `ds:KeyInfo` element to be used. If some information is omitted, the **application** MUST supply it.
2. Locate the data encryption key according to the `ds:KeyInfo` element, which may contain one or more children elements. These children have no implied processing order. If the data encryption key is encrypted, locate the corresponding key to decrypt it. (This may be a recursive step as the key-encryption key may itself be encrypted.) Or, one might retrieve the data encryption key from a local store using the provided attributes or implicit binding.
3. Decrypt the data contained in the `CipherData` element.
   1. If a `CipherValue` child element is present, then the associated text value is retrieved and base64 decoded so as to obtain the encrypted octet sequence.
   2. If a `CipherReference` child element is present, the URI and transforms (if any) are used to retrieve the encrypted octet sequence.
   3. The encrypted octet sequence is decrypted using the algorithm/parameters and key value already determined from steps 1 and 2.
4. Process decrypted data of `Type` 'element' or element 'content'.
   1. The cleartext octet sequence obtained in step 3 is interpreted as UTF-8 encoded character data.

2. The **decryptor** MUST be able to return the value of `Type` and the UTF-8 encoded XML character data. The **decryptor** is NOT REQUIRED to perform validation on the serialized XML.
3. The **decryptor** SHOULD support the ability to replace the `EncryptedData` element with the decrypted 'element' or element 'content' represented by the UTF-8 encoded characters. The **decryptor** is NOT REQUIRED to perform validation on the result of this replacement operation.

   The application supplies the XML document context and identifies the `EncryptedData` element being replaced. If the document into which the replacement is occurring is not UTF-8, the **decryptor** MUST transcode the UTF-8 encoded characters into the target encoding.

5. Process decrypted data if `Type` is unspecified or is *not* 'element' or element 'content'.
   1. The cleartext octet sequence obtained in **Step 3** MUST be returned to the **application** for further processing along with the `Type`, `MimeType`, and `Encoding` attribute values when specified. `MimeType` and `Encoding` are advisory. The `Type` value is normative as it may contain information necessary for the processing or interpration of the data by the application.
   2. Note, this step includes processing data decrypted from an `EncryptedKey`. The cleartext octet sequence represents a key value and is used by the application in decrypting other `EncryptedType` element(s).

4.3 XML Encryption

Encryption and decryption operations are transforms on octets. The **application** is responsible for the marshalling XML such that it can be serialized into an octet sequence, encrypted, decrypted, and be of use to the recipient.

Por ejemplo, si la aplicación desea canonicalizar sus datos o codificar/comprimir los datos en un formato de empaquetado XML, la aplicación necesita ordenar el XML en consecuencia e identificar el tipo resultante mediante el `EncryptedData Type`atributo. La probabilidad de que el descifrado y el procesamiento posterior sean exitosos dependerán del soporte del destinatario para el tipo dado. Además, si se pretende que los datos se procesen antes del cifrado y después del descifrado (por ejemplo, validación de firma XML [XML-DSIG] o una transformación XSLT), la aplicación de cifrado debe tener cuidado de preservar la información necesaria para el éxito de ese proceso.

Para fines de interoperabilidad, DEBEN implementarse los siguientes tipos de modo que una implementación pueda tomar como entrada y producir como salida datos que coincidan con las reglas de producción 39 y 43 de [XML] :

elemento ' http://www.w3.org/2001/04/xmlenc#Element '
"[39] elemento ::= `EmptyElemTag` | Contenido de STag ETag "
contenido ' http://www.w3.org/2001/04/xmlenc#Content '
"[43] contenido ::= `CharData`? ((elemento | Referencia | CDSect | PI | Comentario) `CharData`?)*"

Las siguientes secciones contienen especificaciones para descifrar, reemplazar y serializar contenido XML (es decir, `Type`' elemento ' o elemento ' contenido ') utilizando el modelo de datos [XPath]. Estas secciones no son normativas y son OPCIONALES para los implementadores de esta especificación, pero pueden ser referenciadas normativamente y OBLIGATORIAS para otras especificaciones que requieren un procesamiento consistente para aplicaciones, como [XML-DSIG-Decrypt] .

4.3.1 Una implementación de Decrypt (no normativa)

Donde *P* es el contexto en el que se debe analizar el XML serializado (un nodo de documento o nodo de elemento) y *O* es la secuencia de octetos que representa caracteres codificados en UTF-8 resultantes del paso 4.3 en el *Procesamiento de descifrado* (sección 4.2). *Y* es un conjunto de nodos que representa el contenido descifrado obtenido mediante los siguientes pasos:

1. Sea *C* el **contexto de análisis** de un hijo de *P* , que consta de los siguientes elementos:
   - Prefijo y nombre del espacio de nombres de cada espacio de nombres que está dentro del alcance de *P* .
   - Nombre y valor de cada entidad general que es efectiva para el documento XML que causa *P* .
2. Envuelva el flujo de octetos descifrado *O* en el contexto *C* como se especifica en *Ajuste de texto* .
3. Analice el flujo de octetos envueltos como se describe en *El modelo de procesamiento de referencia* (sección 4.3.3.2) de [Firma XML], lo que da como resultado un conjunto de nodos.
4. *Y* es el conjunto de nodos obtenido al eliminar el nodo raíz, el nodo del elemento envolvente y su conjunto asociado de nodos de atributos y espacios de nombres del conjunto de nodos obtenido en el Paso 3.

4.3.2 Una implementación de descifrado y reemplazo (no normativa)

Donde *X* es el conjunto de nodos [XPath] correspondiente a un documento XML y *e* es un `EncryptedData`nodo de elemento en *X* .

1. *Z* es un conjunto de nodos [XPath] idéntico a X excepto donde el nodo de elemento *e* es un `EncryptedData`tipo de elemento. En ese caso:
    1. Descifre *e* en el contexto de su nodo principal como se especifica en la *Implementación de descifrado* (sección 4.3.1), lo que produce *Y* , un conjunto de nodos [XPath].
    2. Incluya *Y* en lugar de *e* y sus descendientes en *X* . Dado que [XPath] no define métodos para reemplazar conjuntos de nodos de diferentes documentos, el resultado DEBE ser equivalente a reemplazar e con el flujo de octetos resultante de su descifrado en la forma serializada de X y *analizar* el documento. Sin embargo, el método real para realizar esta operación queda en manos del implementador.

4.3.3 Serialización de XML (no normativo)

Consideraciones sobre el espacio de nombres predeterminado

En *Cifrado de XML* (sección 4.1, paso 3.1), al serializar un fragmento XML DEBE tenerse especial cuidado con respecto a los espacios de nombres predeterminados. Si los datos se descifran posteriormente en el contexto de un documento XML principal, la serialización puede producir elementos en el espacio de nombres incorrecto. Considere el siguiente fragmento de XML:

```
<Documento xmlns="http://ejemplo.org/">
  <ToBeEncrypted xmlns="" />
</Documento>
```

La serialización del `ToBeEncrypted`fragmento de elemento mediante [XML-C14N] daría como resultado los caracteres " `<ToBeEncrypted></ToBeEncrypted>`" como una secuencia de octetos. El documento cifrado resultante sería:

```
<Documento xmlns="http://ejemplo.org/">
  <EncryptedData xmlns="...">
    <!-- Contiene el cifrado
         "<ToBeEncrypted></ToBeEncrypted>" -->
  </EncryptedData>
</Documento>
```

Descifrar y reemplazar el `EncryptedData`contenido de este documento produciría el siguiente resultado incorrecto:

```
<Documento xmlns="http://ejemplo.org/">
  <Para ser cifrado/>
</Document>
```

This problem arises because most XML serializations assume that the serialized data will be parsed directly in a context where there is no default namespace declaration. Consequently, they do not redundantly declare the empty default namespace with an `xmlns=""`. If, however, the serialized data is parsed in a context where a default namespace declaration is in scope (e.g., the parsing context of a *A Decrypt Implementation* (section 4.3.1)), then it may affect the interpretation of the serialized data.

To solve this problem, a canonicalization algorithm MAY be augmented as follows for use as an XML encryption serializer:

- A default namespace declaration with an empty value (i.e., `xmlns=""`) SHOULD be emitted where it would normally be suppressed by the canonicalization algorithm.

While the result may not be in proper canonical form, this is harmless as the resulting octet stream will not be used directly in a [XML-Signature] signature value computation. Returning to the preceding example with our new augmentation, the `ToBeEncrypted` element would be serialized as follows:

```
<ToBeEncrypted xmlns=""></ToBeEncrypted>
```

When processed in the context of the parent document, this serialized fragment will be parsed and interpreted correctly.

This augmentation can be retroactively applied to an existing canonicalization implementation by canonicalizing each apex node and its descendants from the node set, inserting `xmlns=""` at the appropriate points, and concatenating the resulting octet streams.

XML Attribute Considerations

Similar attention between the relationship of a fragment and the context into which it is being inserted should be given to the xml:base, xml:lang, and xml:space attributes as mentioned in the *Security Considerations* of [XML-exc-C14N]. For example, if the element:

```
<Bongo href="example.xml"/>
```

is taken from a context and serialized with no xml:base [XML-Base] attribute and parsed in the context of the element:

```
<Baz xml:base="http://example.org/"/>
```

the result will be:

```
<Baz xml:base="http://example.org/"><Bongo href="example.xml"/></Baz>
```

Bongo's href is subsequently interpreted as "http://example.org/example.xml". If this is not the correct URI, Bongo should have been serialized with its own xml:base attribute.

Desafortunadamente, la recomendación de que se emita un valor vacío para separar el espacio de nombres predeterminado del fragmento del contexto en el que se inserta no se puede realizar para los atributos xml:basey xml:space. (*El error 41* de la errata de especificación XML 1.0 de segunda edición aclara que un valor de cadena vacío del atributo xml:lang *se* considera como si "no hubiera información de idioma disponible, como si xml:langno se hubiera especificado".) La interpretación de un valor vacío para los atributos xml:baseo xml:spaceno están definidos o mantienen el valor contextual. En consecuencia, las aplicaciones DEBEN garantizar (1) que los fragmentos que se van a cifrar no dependan de atributos XML, o (2) si son dependientes y se pretende que el documento resultante sea válido [XML], la definición del fragmento *permite* la presencia del atributos y que los atributos tengan valores no vacíos.

4.3.4 Ajuste de texto (no normativo)

Esta sección especifica el proceso para ajustar texto en un contexto de análisis determinado. El proceso se basa en la propuesta de Richard Tobin [Tobin] para construir el conjunto de información [XML-Infoset] de una entidad externa.

El proceso consta de los siguientes pasos:

1. Si el contexto de análisis contiene entidades generales, emita una declaración de tipo de documento que proporcione declaraciones de entidades.
2. Emita una dummyetiqueta de inicio de elemento con atributos de declaración de espacio de nombres que declaren todos los espacios de nombres en el contexto de análisis.
3. Emitir el texto.
4. Emitir una dummyetiqueta final de elemento.

En los pasos anteriores, la declaración del tipo de documento y dummy las etiquetas de elementos DEBEN codificarse en UTF-8.

Considere el siguiente documento que contiene un EncryptedData elemento:

```
<!DOCTYPE Documento [
  <!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<Documento xmlns="http://ejemplo.org/">
  <foo:Cuerpo xmlns:foo="http://example.org/foo">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                   Escriba=" http://www.w3.org/2001/04/xmlenc#Element ">
      ...
    </EncryptedData>
  </foo:Cuerpo>
</Documento>
```

Si el EncryptedDataelemento se alimenta y se descifra en el texto " <One><foo:Two/></One>", entonces el formato ajustado es el siguiente:

```
<!DOCTYPE ficticio [
  <!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<dummy xmlns="http://ejemplo.org/"
       xmlns:foo="http://example.org/foo"><Uno><foo:Two/></Uno></dummy>
```

5. Algoritmos

Esta sección analiza los algoritmos utilizados con la especificación de cifrado XML. Las entradas contienen el identificador que se utilizará como valor del `Algorithm`atributo del `EncryptionMethod`elemento u otro elemento que represente el rol del algoritmo, una referencia a la especificación formal, definiciones para la representación de claves y los resultados de las operaciones criptográficas cuando corresponda, y información general. comentarios de aplicabilidad.

5.1 Identificadores de algoritmos y requisitos de implementación

Todos los algoritmos enumerados a continuación tienen parámetros implícitos según su función. Por ejemplo, los datos que se van a cifrar o descifrar, el material de claves y la dirección de operación (cifrado o descifrado) de los algoritmos de cifrado. Cualquier parámetro adicional explícito de un algoritmo aparece como elementos de contenido dentro del elemento. Dichos elementos secundarios de parámetros tienen nombres de elementos descriptivos, que frecuentemente son específicos del algoritmo, y DEBEN estar en el mismo espacio de nombres que esta especificación de cifrado XML, la especificación de firma XML o en un espacio de nombres específico del algoritmo. Un ejemplo de un parámetro tan explícito podría ser un nonce (cantidad única) proporcionado a un algoritmo de acuerdo clave.

Esta especificación define un conjunto de algoritmos, sus URI y requisitos de implementación. Los niveles de requisitos especificados, como "REQUERIDO" u "OPCIONAL", se refieren a la implementación, no al uso. Además, el mecanismo es extensible y se pueden utilizar algoritmos alternativos.

Tabla de algoritmos

La siguiente tabla enumera las categorías de algoritmos. Dentro de cada categoría, se proporciona un nombre breve, el nivel de requisito de implementación y un URI de identificación para cada algoritmo.

Cifrado de bloques

1. TRIPLEDES REQUERIDOS
   http://www.w3.org/2001/04/xmlenc#tripledes-cbc
2. AES-128 REQUERIDO
   http://www.w3.org/2001/04/xmlenc#aes128-cbc
3. AES-256 REQUERIDO
   http://www.w3.org/2001/04/xmlenc#aes256-cbc
4. OPCIONAL AES-192
   http://www.w3.org/2001/04/xmlenc#aes192-cbc

Cifrado de flujo

1. none
   A continuación se proporcionan sintaxis y recomendaciones para admitir algoritmos especificados por el usuario.

Transporte clave

1. REQUERIDO RSA-v1.5
   http://www.w3.org/2001/04/xmlenc#rsa-1_5
2. REQUERIDO RSA-OAEP
   http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p

Acuerdo clave

1. OPCIONAL Diffie-Hellman
   http://www.w3.org/2001/04/xmlenc#dh

Envoltura de clave simétrica

1. TRIPLEDES REQUERIDOS KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-tripledes
2. REQUERIDO AES-128 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes128
3. REQUERIDO AES-256 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes256
4. OPCIONAL AES-192 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes192

Resumen del mensaje

1. SHA1 REQUERIDO
   http://www.w3.org/2000/09/xmldsig#sha1
2. RECOMMENDED SHA256
   http://www.w3.org/2001/04/xmlenc#sha256
3. OPTIONAL SHA512
   http://www.w3.org/2001/04/xmlenc#sha512
4. OPTIONAL RIPEMD-160
   http://www.w3.org/2001/04/xmlenc#ripemd160

Message Authentication

1. RECOMMENDED XML Digital Signature
   http://www.w3.org/2000/09/xmldsig#

Canonicalization

1. OPTIONAL Canonical XML (omits comments)
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315
2. OPTIONAL Canonical XML with Comments
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
3. OPTIONAL Exclusive XML Canonicalization (omits comments)
   http://www.w3.org/2001/10/xml-exc-c14n#
4. OPTIONAL Exclusive XML Canonicalization with Comments
   http://www.w3.org/2001/10/xml-exc-c14n#WithComments

Encoding

1. REQUIRED base64
   http://www.w3.org/2000/09/xmldsig#base64

5.2 Block Encryption Algorithms

Block encryption algorithms are designed for encrypting and decrypting data in fixed size, multiple octet blocks. Their identifiers appear as the value of the `Algorithm` attributes of `EncryptionMethod` elements that are children of `EncryptedData`.

Block encryption algorithms take, as implicit arguments, the data to be encrypted or decrypted, the keying material, and their direction of operation. For all of these algorithms specified below, an initialization vector (IV) is required that is encoded with the cipher text. For user specified block encryption algorithms, the IV, if any, could be specified as being with the cipher data, as an algorithm content element, or elsewhere.

The IV is encoded with and before the cipher text for the algorithms below for ease of availability to the decryption code and to emphasize its association with the cipher text. Good cryptographic practice requires that a different IV be used for every encryption.

Padding

Since the data being encrypted is an arbitrary number of octets, it may not be a multiple of the block size. This is solved by padding the plain text up to the block size before encryption and unpadding after decryption. The padding algorithm is to calculate the smallest non-zero number of octets, say `N`, that must be suffixed to the plain text to bring it up to a multiple of the block size. We will assume the block size is `B` octets so `N` is in the range of 1 to `B`. Pad by suffixing the plain text with `N-1` arbitrary pad bytes and a final byte whose value is `N`. On decryption, just take the last byte and, after sanity checking it, strip that many bytes from the end of the decrypted cipher text.

For example, assume an 8 byte block size and plain text of `0x616263`. The padded plain text would then be `0x616263??????????05` where the "??" bytes can be any value. Similarly, plain text of `0x2122232425262728` would be padded to `0x2122232425262728??????????????08`.

5.2.1 Triple DES

Identifier:
http://www.w3.org/2001/04/xmlenc#tripledes-cbc (REQUIRED)

ANSI X9.52 [TRIPLEDES] specifies three sequential FIPS 46-3 [DES] operations. The XML Encryption TRIPLEDES consists of a DES encrypt, a DES decrypt, and a DES encrypt used in the Cipher Block Chaining (CBC) mode with 192 bits of key and a 64 bit Initialization Vector (IV). Of the key bits, the first 64 are used in the first DES operation, the second 64 bits in the middle DES operation, and the third 64 bits in the last DES operation.

Note: Each of these 64 bits of key contain 56 effective bits and 8 parity bits. Thus there are only 168 operational bits out of the 192 being transported for a TRIPLEDES key. (Depending on the criterion used for analysis, the effective strength of the key may be thought to be 112 bits (due to meet in the middle attacks) or even less.)

The resulting cipher text is prefixed by the IV. If included in XML output, it is then base64 encoded. An example TRIPLEDES EncryptionMethod is as follows:

```
<EncryptionMethod
 Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
```

5.2.2 AES

Identifier:
http://www.w3.org/2001/04/xmlenc#aes128-cbc (REQUIRED)
http://www.w3.org/2001/04/xmlenc#aes192-cbc (OPTIONAL)
http://www.w3.org/2001/04/xmlenc#aes256-cbc (REQUIRED)

[AES] is used in the Cipher Block Chaining (CBC) mode with a 128 bit initialization vector (IV). The resulting cipher text is prefixed by the IV. If included in XML output, it is then base64 encoded. An example AES EncryptionMethod is as follows:

```
<EncryptionMethod
 Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
```

5.3 Stream Encryption Algorithms

Simple stream encryption algorithms generate, based on the key, a stream of bytes which are XORed with the plain text data bytes to produce the cipher text on encryption and with the cipher text bytes to produce plain text on decryption. They are normally used for the encryption of data and are specified by the value of the `Algorithm` attribute of the `EncryptionMethod` child of an `EncryptedData` element.

NOTE: It is critical that each simple stream encryption key (or key and initialization vector (IV) if an IV is also used) be used once only. If the same key (or key and IV) is ever used on two messages then, by XORing the two cipher texts, you can obtain the XOR of the two plain texts. This is usually very compromising.

No specific stream encryption algorithms are specified herein but this section is included to provide general guidelines.

Stream algorithms typically use the optional `KeySize` explicit parameter. In cases where the key size is not apparent from the algorithm URI or key source, as in the use of key agreement methods, this parameter sets the key size. If the size of the key to be used is apparent and disagrees with the `KeySize` parameter, an error MUST be returned. Implementation of any stream algorithms is optional. The schema for the KeySize parameter is as follows:

```
Schema Definition:

<simpleType name='KeySizeType'>
  <restriction base="integer"/>
</simpleType>
```

5.4 Key Transport

Key Transport algorithms are public key encryption algorithms especially specified for encrypting and decrypting keys. Their identifiers appear as `Algorithm` attributes to `EncryptionMethod` elements that are children of `EncryptedKey`. `EncryptedKey` is in turn the child of a `ds:KeyInfo` element. The type of key being transported, that is to say the algorithm in which it is planned to use the transported key, is given by the `Algorithm` attribute of the `EncryptionMethod` child of the `EncryptedData` or `EncryptedKey` parent of this `ds:KeyInfo` element.

(Key Transport algorithms may optionally be used to encrypt data in which case they appear directly as the `Algorithm` attribute of an `EncryptionMethod` child of an `EncryptedData` element. Because they use public key algorithms directly, Key Transport algorithms are not efficient for the transport of any amounts of data significantly larger than symmetric keys.)

The RSA v1.5 Key Transport algorithm given below are those used in conjunction with TRIPLEDES and the Cryptographic Message Syntax (CMS) of S/MIME [CMS-Algorithms]. The RSA v2 Key Transport algorithm given below is that used in conjunction with AES and CMS [AES-WRAP].

5.4.1 RSA Version 1.5

Identifier:
 http://www.w3.org/2001/04/xmlenc#rsa-1_5 (REQUIRED)

The RSAES-PKCS1-v1_5 algorithm, specified in RFC 2437 [PKCS1], takes no explicit parameters. An example of an RSA Version 1.5 `EncryptionMethod` element is:

```
<EncryptionMethod
  Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
```

The `CipherValue` for such an encrypted key is the base64 [MIME] encoding of the octet string computed as per RFC 2437 [PKCS1, section 7.2.1: Encryption operation]. As specified in the EME-PKCS1-v1_5 function RFC 2437 [PKCS1, section 9.1.2.1], the value input to the key transport function is as follows:

```
CRYPT ( PAD ( KEY ))
```

where the padding is of the following special form:

```
02 | PS* | 00 | key
```

where "|" is concatenation, "02" and "00" are fixed octets of the corresponding hexadecimal value, PS is a string of strong pseudo-random octets [RANDOM] at least eight octets long, containing no zero octets, and long enough that the value of the quantity being CRYPTed is one octet shorter than the RSA modulus, and "key" is the key being transported. The key is 192 bits for TRIPLEDES and 128, 192, or 256 bits for AES. Support of this key transport algorithm for transporting 192 bit keys is MANDATORY to implement. Support of this algorithm for transporting other keys is OPTIONAL. RSA-OAEP is RECOMMENDED for the transport of AES keys.

The resulting base64 [MIME] string is the value of the child text node of the `CipherData` element, e.g.

```
<CipherData> IWijxQjUrcXBYoCei4QxjWo9Kg8D3p9tlWoT4
   t0/gyTE96639In0FZFY2/rvP+/bMJ01EArmKZsR5VW3rwoPxw=
</CipherData>
```

5.4.2 RSA-OAEP

Identifier:
 http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p (REQUIRED)

The RSAES-OAEP-ENCRYPT algorithm, as specified in RFC 2437 [PKCS1], takes three parameters. The two user specified parameters are a MANDATORY message digest function and an OPTIONAL encoding octet string `OAEPparams`. The message digest function is indicated by the `Algorithm` attribute of a child `ds:DigestMethod` element and the mask generation function, the third parameter, is always MGF1 with SHA1 (mgf1SHA1Identifier). Both the message digest and mask generation functions are used in the EME-OAEP-ENCODE operation as part of RSAES-OAEP-ENCRYPT. The encoding octet string is the base64 decoding of the content of an optional `OAEPparams` child element . If no `OAEPparams` child is provided, a null string is used.

```
Schema Definition:
     <!-- use these element types as children of EncryptionMethod
          when used with RSA-OAEP -->
     <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
     <element ref='ds:DigestMethod' minOccurs='0'/>
```

An example of an RSA-OAEP element is:

```
<EncryptionMethod
   Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
  <OAEPparams> 9lWu3Q== </OAEPparams>
  <ds:DigestMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<EncryptionMethod>
```

The `CipherValue` for an RSA-OAEP encrypted key is the base64 [MIME] encoding of the octet string computed as per RFC 2437 [PKCS1, section 7.1.1: Encryption operation]. As described in the EME-OAEP-ENCODE function RFC 2437 [PKCS1, section 9.1.1.1], the value input to the key transport function is calculated using the message digest function and string specified in the `DigestMethod` and `OAEPparams` elements and using the mask generator function MGF1 (with SHA1) specified in RFC 2437. The desired output length for EME-OAEP-ENCODE is one byte shorter than the RSA modulus.

The transported key size is 192 bits for TRIPLEDES and 128, 192, or 256 bits for AES. Implementations MUST implement RSA-OAEP for the transport of 128 and 256 bit keys. They MAY implement RSA-OAEP for the transport of other keys.

5.5 Key Agreement

A Key Agreement algorithm provides for the derivation of a shared secret key based on a shared secret computed from certain types of compatible public keys from both the sender and the recipient. Information from the originator to determine the secret is indicated by an optional `OriginatorKeyInfo` parameter child of an `AgreementMethod` element while that associated with the recipient is indicated by an optional `RecipientKeyInfo`. A shared key is derived from this shared secret by a method determined by the Key Agreement algorithm.

Note: XML Encryption does not provide an on-line key agreement negotiation protocol. The `AgreementMethod` element can be used by the originator to identify the keys and computational procedure that were used to obtain a shared encryption key. The method used to obtain or select the keys or algorithm used for the agreement computation is beyond the scope of this specification.

The `AgreementMethod` element appears as the content of a `ds:KeyInfo` since, like other `ds:KeyInfo` children, it yields a key. This `ds:KeyInfo` is in turn a child of an `EncryptedData` or `EncryptedKey` element. The `Algorithm` attribute and `KeySize` child of the `EncryptionMethod` element under this `EncryptedData` or `EncryptedKey` element are implicit parameters to the key agreement computation. In cases where this `EncryptionMethod` algorithm URI is insufficient to determine the key length, a `KeySize` MUST have been included. In addition, the sender may place a `KA-Nonce` element under `AgreementMethod` to assure that different keying material is generated even for repeated agreements using the same sender and recipient public keys. For example:

```
<EncryptedData>
  <EncryptionMethod Algorithm="Example:Block/Alg"
    <KeySize>80</KeySize>
  </EncryptionMethod>
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <AgreementMethod Algorithm="example:Agreement/Algorithm">
      <KA-Nonce>Zm9v</KA-Nonce>
      <ds:DigestMethod
      Algoritmo="http://www.w3.org/2001/04/xmlenc#sha1"/>
     <Información de clave del originador>
        <ds:ValorClave>....</ds:ValorClave>
      </OriginatorKeyInfo>
      <Información de clave del destinatario>
        <ds:ValorClave>....</ds:ValorClave>
      </RecipientKeyInfo>
    </Método de acuerdo>
  </ds:Información clave>
  <CipherData>...</CipherData>
</EncryptedData>
```

Si la clave acordada se utiliza para envolver una clave, en lugar de datos como se indica arriba, `AgreementMethod`aparecerá dentro de `ds:KeyInfo`un `EncryptedKey`elemento.

El esquema `AgreementMethod`es el siguiente:

```
Definición del esquema:

<elemento nombre="AgreementMethod" type="xenc:AgreementMethodType"/>
<complexType nombre="AgreementMethodType" mixto="verdadero">
  <secuencia>
    <elemento nombre="KA-Nonce" minOccurs="0" tipo="base64Binary"/>
    <!-- <elemento ref="ds:DigestMethod" minOccurs="0"/> -->
    <cualquier espacio de nombre="##other" minOccurs="0" maxOccurs="ilimitado"/>
    <elemento nombre="OriginatorKeyInfo" minOccurs="0"
             tipo="ds:KeyInfoType"/>
    <elemento nombre="RecipientKeyInfo" minOccurs="0"
             tipo="ds:KeyInfoType"/>
  </secuencia>
  <atributo nombre="Algoritmo" tipo="cualquierURI" uso="requerido"/>
</tipocomplejo>
```

5.5.1 Valores clave de Diffie-Hellman

Identificador:
    http://www.w3.org/2001/04/xmlenc#DHKeyValue (OPCIONAL)

Las claves Diffie-Hellman pueden aparecer directamente dentro de `KeyValue` los elementos u obtenerse mediante `ds:RetrievalMethod`recuperaciones, además de aparecer en certificados y similares. El identificador anterior se puede utilizar como el valor del `Type`atributo de `Reference`o `ds:RetrievalMethod`elementos.

Como se especifica en [ESDH], una clave pública DH consta de hasta seis cantidades, dos números primos grandes p y q, un "generador" g, la clave pública y los parámetros de validación "seed" y "pgenCounter". Estos se relacionan de la siguiente manera: La clave pública = (g**x mod p) donde x es la clave privada correspondiente;

p = j*q + 1 donde j >= 2. "seed" y "pgenCounter" son opcionales y se pueden utilizar para determinar si la clave Diffie-Hellman se ha generado de conformidad con el algoritmo especificado en [ESDH] . Debido a que los números primos y el generador se pueden compartir de forma segura entre muchas claves DH, es posible que se conozcan desde el entorno de la aplicación y son opcionales. El esquema para a `DHKeyValue`es el siguiente:

```
 Schema:

<element name="DHKeyValue" type="xenc:DHKeyValueType"/>
<complexType name="DHKeyValueType">
   <sequence>
      <sequence minOccurs="0">
         <element name="P" type="ds:CryptoBinary"/>
         <element name="Q" type="ds:CryptoBinary"/>
         <element name="Generator"type="ds:CryptoBinary"/>
      </sequence>
      <element name="Public" type="ds:CryptoBinary"/>
      <sequence minOccurs="0">
         <element name="seed" type="ds:CryptoBinary"/>
         <element name="pgenCounter" type="ds:CryptoBinary"/>
      </sequence>
   </sequence>
</complexType>
```

5.5.2 Acuerdo clave Diffie-Hellman

Identificador:

http://www.w3.org/2001/04/xmlenc#dh (OPCIONAL)

El protocolo de acuerdo de claves Diffie-Hellman (DH) [ESDH] implica la derivación de información secreta compartida basada en claves DH compatibles del remitente y el destinatario. Dos claves públicas DH son compatibles si tienen el mismo número principal y generador. Si, para el segundo, `Y = g**y mod p`entonces las dos partes pueden calcular el secreto compartido `ZZ = ( g**(x*y) mod p )`aunque cada una conozca sólo su propia clave privada y la clave pública de la otra parte. Los bytes cero iniciales DEBEN mantenerse para `ZZ`que tengan la misma longitud, en bytes, que `p`. El tamaño `p`DEBE ser de al menos 512 bits y `g`al menos 160 bits. Existen muchas otras consideraciones de seguridad complejas en la selección de `g`, `p`y aleatorio `x`como se describe en [ESDH].

El acuerdo clave Diffie-Hellman es opcional de implementar. Un ejemplo de un `AgreementMethod`elemento DH es el siguiente:

```
<Método de acuerdo
    Algoritmo="http://www.w3.org/2001/04/xmlenc#dh"
    ds:xmlns="http://www.w3.org/2000/09/xmldsig#">
  <KA-Nonce>Zm9v</KA-Nonce>
  <ds:Método de resumen
   Algoritmo="http://www.w3.org/2000/09/xmldsig#sha1"/>
  <Información de clave del originador>
    <ds:X509Data><ds:X509Certificado>
      ...
    </ds:X509Certificate></ds:X509Data>
  </OriginatorKeyInfo>
  <RecipientKeyInfo><ds:KeyValue>
    ...
  </ds:KeyValue></RecipientKeyInfo>
</Método de acuerdo>
```

Supongamos que el secreto compartido de Diffie-Hellman es la secuencia de octetos `ZZ`. El material de claves compartido necesario se calculará de la siguiente manera:

```
Material de codificación = KM(1) | KM(2) | ...
```

donde "|" es la concatenación de flujo de bytes y

```
KM(contador) = DigestAlg ( ZZ | contador | EncryptionAlg |
              KA-Nonce | Tamaño de clave)
```

`DigestAlg`

El algoritmo de resumen de mensajes especificado por el `DigestMethod`hijo de `AgreementMethod`.

`EncryptionAlg`

El URI del algoritmo de cifrado, incluidos los posibles algoritmos de ajuste de claves, en los que se utilizará el material de claves derivado ("Ejemplo: Bloque/Alg" en el ejemplo anterior), no el URI del algoritmo de acuerdo. Este es el valor del `Algorithm` atributo del `EncryptionMethod`hijo de `EncryptedData`o `EncryptedKey`abuelo de `AgreementMethod`.

`KA-Nonce`

Base64 decodifica el contenido del `KA-Nonce`hijo de `AgreementMethod`, si está presente. Si el `KA-Nonce` elemento está ausente, es nulo.

`Counter`
: Un contador de un byte que comienza en uno y se incrementa en uno. Se expresa como dos dígitos hexadecimales donde las letras de la A a la F están en mayúsculas.

`KeySize`
: El tamaño en bits de la clave que se derivará del secreto compartido como la cadena UTF-8 para el entero decimal correspondiente con solo dígitos en la cadena y sin ceros a la izquierda. Para algunos algoritmos, el tamaño de la clave es inherente al URI. Para otros, como la mayoría de los cifrados de flujo, se debe proporcionar explícitamente.

`(KM(1))`Por ejemplo, el cálculo inicial para el ejemplo `EncryptionMethod`de [Acuerdo de clave](#) (sección 5.5) sería el siguiente, donde el valor del contador binario de un byte de 1 está representado por la secuencia UTF-8 de dos caracteres `01`, `ZZ`es el secreto compartido y " `foo`" es la decodificación base64 de " `Zm9v`".

```
SHA-1 (ZZ01Ejemplo:Bloque/Algfoo80)
```

Suponiendo que así `ZZ`sea `0xDEADBEEF`, eso sería

```
SHA-1 (0xDEADBEEF30314578616D706C653A426C6F636B2F416C67666F6F3830)
```

cuyo valor es

```
0x534C9B8C4ABDCB50038B42015A181711068B08C1
```

Cada aplicación de `DigestAlg`para valores sucesivos de `Counter`producirá un número adicional de bytes de material de claves. De la cadena concatenada de uno o más `KM`, se toman suficientes bytes iniciales para satisfacer la necesidad de una clave real y el resto se descarta. Por ejemplo, si `DigestAlg`SHA-1 produce 20 octetos de hash, entonces para AES de 128 bits `KM(1)`se tomarían los primeros 16 bytes y se descartarían los 4 bytes restantes. `KM(1)`Para AES de 256 bits, se tomarían todos los sufijos con los primeros 12 bytes de KM(2) y `KM(2)` se descartarían los 8 bytes restantes.

5.6 Ajuste de clave simétrica

Los algoritmos Symmetric Key Wrap son algoritmos de cifrado de claves secretas compartidas especialmente especificados para cifrar y descifrar claves simétricas. Sus identificadores aparecen como `Algorithm`valores de atributos de `EncryptionMethod`elementos que son hijos de `EncryptedKey`los cuales, a su vez, `ds:KeyInfo` son hijos de `EncryptedData`u otros `EncryptedKey`. El tipo de clave que se encapsula se indica mediante el `Algorithm`atributo de `EncryptionMethod`hijo del padre del `ds:KeyInfo`abuelo del que `EncryptionMethod`especifica el algoritmo de encapsulación de clave simétrica.

5.6.1 Suma de comprobación de clave CMS

Algunos algoritmos de ajuste de claves utilizan una suma de comprobación de claves como se define en CMS [[CMS-Wrap](#)]. El algoritmo que proporciona un valor de verificación de integridad para la clave que se empaqueta es:

1. Calcule el hash SHA-1 de 20 octetos en la clave que se está empaquetando.
2. Utilice los primeros 8 octetos de este hash como valor de suma de comprobación.

5.6.2 Envoltura de claves Triple DES de CMS

Identificadores y requisitos:
: http://www.w3.org/2001/04/xmlenc#kw-tripledes (REQUERIDO)

Las implementaciones de cifrado XML DEBEN admitir la envoltura TRIPLEDES de claves de 168 bits y, opcionalmente, pueden admitir la envoltura TRIPLEDES de otras claves.

Un ejemplo de un `EncryptionMethod`elemento Key Wrap TRIPLEDES es el siguiente:

```
<EncryptionMethod
 Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"/>
```

El siguiente algoritmo envuelve (cifra) una clave (la clave envuelta, WK) bajo una clave de cifrado de clave TRIPLEDES (KEK) adoptada de [[Algoritmos CMS](#)]:

1. Representa la clave que se encapsula como una secuencia de octetos. Si se trata de una clave TRIPLEDES, son 24 octetos (192 bits) con un bit de paridad impar como bit inferior de cada octeto.

2. Calcule la suma de verificación de la clave CMS (sección 5.6.1), llame a esto CKS.
3. Sea `WKCKS = WK || CKS`, donde || es la concatenación.
4. Genere 8 octetos aleatorios [RANDOM] y llame a esto IV.
5. Cifre WKCKS en modo CBC utilizando KEK como clave y IV como vector de inicialización. Llame a los resultados TEMP1.
6. Deja `TEMP2 = IV || TEMP1`.
7. Invierta el orden de los octetos `TEMP2`y llame al resultado `TEMP3`.
8. Cifre `TEMP3`en modo CBC utilizando `KEK`y un vector de inicialización de `0x4adda22c79e82105`. El texto cifrado resultante es el resultado deseado. Tiene una longitud de 40 octetos si se encapsula una clave de 168 bits.

El siguiente algoritmo desenvuelve (descifra) una clave adoptada de [Algoritmos CMS]:

1. Compruebe si la longitud del texto cifrado es razonable dado el tipo de clave. Debe tener 40 bytes para una clave de 168 bits y 32, 40 o 48 bytes para una clave de 128, 192 o 256 bits. Si la longitud no es compatible o es inconsistente con el algoritmo para el cual está destinada la clave, se devuelve un error.
2. Descifre el texto cifrado con TRIPLEDES en modo CBC utilizando `KEK`y un vector de inicialización (IV) de `0x4adda22c79e82105`. Llame a la salida `TEMP3`.
3. Invierta el orden de los octetos en TEMP3 y llame al resultado `TEMP2`.
4. Descomponga TEMP2 en IV, los primeros 8 octetos y `TEMP1`los octetos restantes.
5. Descifre TEMP1 usando TRIPLEDES en modo CBC usando `KEK` y el IV que se encuentra en el paso anterior. Llame al resultado `WKCKS`.
6. Descomponer `WKCKS`. `CKS`son los últimos 8 octetos y `WK`, la clave envuelta, son los octetos anteriores al `CKS`.
7. Calcule una suma de verificación de clave CMS (sección 5.6.1) sobre `WK`y compárela con la `CKS` extraída en el paso anterior. Si no son iguales, devuelve error.
8. `WK`es la clave empaquetada, ahora extraída para usarla en el descifrado de datos.

5.6.3 Ajuste de clave AES

Identificadores y requisitos:

http://www.w3.org/2001/04/xmlenc#kw-aes128 (REQUERIDO)
http://www.w3.org/2001/04/xmlenc#kw-aes192 (OPCIONAL)
http://www.w3.org/2001/04/xmlenc#kw-aes256 (REQUERIDO)

La implementación del ajuste de claves AES se describe a continuación, según lo sugerido por NIST. Proporciona confidencialidad e integridad. Este algoritmo se define sólo para entradas que sean múltiplos de 64 bits. La información incluida no tiene por qué ser en realidad una clave. El algoritmo es el mismo independientemente del tamaño de la clave AES utilizada en el empaquetado, denominada clave de cifrado de clave o `KEK`. Los requisitos de implementación se indican a continuación.

Clave de cifrado de clave AES de 128 bits
SE REQUIERE la implementación de envoltura de claves de 128 bits.
Envoltura de otros tamaños de llaves OPCIONAL.
Clave de cifrado de clave AES de 192 bits
Todo el soporte es OPCIONAL.
Clave de cifrado de clave AES de 256 bits
SE REQUIERE la implementación de envoltura de claves de 256 bits.
Envoltura de otros tamaños de llaves OPCIONAL.

Supongamos que los datos que se van a empaquetar constan de `N`bloques de datos de 64 bits denominados `P(1)`, `P(2)`, `P(3)`... `P(N)`. El resultado del ajuste serán `N+1`bloques de 64 bits denominados `C(0)`, `C(1)`, `C(2)`, ... `C(N)`. La clave de cifrado de claves está representada por `K`. Suponga números enteros `i`, `j`y `t`un registro intermedio de 64 bits `A`, un registro de 128 bits `B`y una matriz de cantidades de 64 bits `R(1)`hasta `R(N)`.

"|" representa la concatenación, entonces `x|y`, donde `x` y `y`y cantidades de 64 bits, es la cantidad de 128 bits `x`en los bits más significativos y `y`en los bits menos significativos. `AES(K)enc(x)`es la operación de AES que cifra la cantidad de 128 bits `x`bajo la clave `K`. `AES(K)dec(x)`es la opción de descifrado correspondiente. `XOR(x,y)`es el exclusivo bit a bit o de `x`y `y`. `MSB(x)`y `LSB(y)`son los 64 bits más significativos y los 64 bits menos significativos de `x`e`y` respectivamente.

Si `N`es 1, se realiza una única operación AES para envolver o desenvolver. Si es `N>1`, entonces `6*N`se realizan operaciones AES para envolver o desenvolver.

El algoritmo de ajuste de claves es el siguiente:

1. Si `N`es `1`:
   - `B=AES(K)enc(0xA6A6A6A6A6A6A6A6|P(1))`
   - `C(0)=MSB(B)`
   - `C(1)=LSB(B)`

   Si es así `N>1`, realice los siguientes pasos:

2. Inicializar variables:
   - Establecer `Aen0xA6A6A6A6A6A6A6A6`
   - Para , `i=1_N`
     `R(i)=P(i)`
3. Calcular valores intermedios:
   - Para , `j=0_5`
     - Para , `i=1_N`
       `t= i + j*N`
       `B=AES(K)enc(A|R(i))`
       `A=XOR(t,MSB(B))`
       `R(i)=LSB(B)`
4. Salida de los resultados:
   - Colocar`C(0)=A`
   - Para , `i=1_N`
     `C(i)=R(i)`

El algoritmo de desenvolvimiento de claves es el siguiente:

1. Si `N`es `1`:
   - `B=AES(K)dec(C(0)|C(1))`
   - `P(1)=LSB(B)`
   - Si `MSB(B)`es así `0xA6A6A6A6A6A6A6A6`, devuelve el éxito. De lo contrario, devolverá un error de falla de verificación de integridad.

   Si `N`>1, realice los siguientes pasos:
2. Inicialice las variables:
   - `A=C(0)`
   - Para , `i=1_N`
     `R(i)=C(i)`
3. Calcular valores intermedios:
   - Para , `j=5_0`
     - Para , `i=N_1`
       `t= i + j*N`
       `B=AES(K)dec(XOR(t,A)|R(i))`
       `A=MSB(B)`
       `R(i)=LSB(B)`
4. Salida de los resultados:
   - Para , `i=1_N`
     `P(i)=R(i)`
   - Si `A`es así `0xA6A6A6A6A6A6A6A6`, devuelve el éxito. De lo contrario, devolverá un error de falla de verificación de integridad.

Por ejemplo, envolver los datos `0x00112233445566778899AABBCCDDEEFF`con produce `KEK` `0x000102030405060708090A0B0C0D0E0F`el texto cifrado de `0x1FA68B0A8112B447`,, `0xAEF34BD8FB5A7B82`. `0x9D3E862371D2CFE5`

5.7 Resumen de mensajes

Los algoritmos de resumen de mensajes se pueden utilizar `AgreementMethod`como parte de la derivación de claves, dentro del cifrado RSA-OAEP como función hash y en conexión con el método del código de autenticación de mensajes HMAC como se describe en [XML-DSIG].)

5.7.1 SHA1

Identificador:
http://www.w3.org/2000/09/xmldsig#sha1 (REQUERIDO)

El algoritmo SHA-1 [SHA] no toma parámetros explícitos. Un ejemplo de un `DigestMethod`elemento SHA-1 es:

```
<Método de resumen
 Algoritmo="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

Un resumen SHA-1 es una cadena de 160 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 20 octetos. Por ejemplo, el `DigestValue`elemento para el resumen del mensaje:

```
A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D
```

del Apéndice A del estándar SHA-1 sería:

```
<DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</DigestValue>
```

5.7.2 SHA256

Identificador:
http://www.w3.org/2001/04/xmlenc#sha256 (RECOMENDADO)

El algoritmo SHA-256 [SHA] no toma parámetros explícitos. `DigestMethod`Un ejemplo de un elemento SHA-256 es:

```
<DigestMethod
 Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
```

Un resumen SHA-256 es una cadena de 256 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 32 octetos.

5.7.3 SHA512

Identificador:
http://www.w3.org/2001/04/xmlenc#sha512 (OPCIONAL)

El algoritmo SHA-512 [SHA] no toma parámetros explícitos. `DigestMethod`Un ejemplo de un elemento SHA-512 es:

```
<Método de resumen
 Algoritmo="http://www.w3.org/2001/04/xmlenc#sha512"/>
```

Un resumen SHA-512 es una cadena de 512 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 64 octetos.

5.7.4 RIPEMD-160

Identificador:
http://www.w3.org/2001/04/xmlenc#ripemd160 (OPCIONAL)

El algoritmo RIPEMD-160 [RIPEMD-160] no toma parámetros explícitos. Un ejemplo de un `DigestMethod` elemento RIPEMD-160 es:

```
<Método de resumen
 Algoritmo="http://www.w3.org/2001/04/xmlenc#ripemd160"/>
```

Un resumen RIPEMD-160 es una cadena de 160 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 20 octetos.

5.8 Autenticación de mensajes

Identificador:
http://www.w3.org/2000/09/xmldsig# (RECOMENDADO)

La firma XML [XML-DSIG] es OPCIONAL para implementar en aplicaciones de cifrado XML. Es la forma recomendada de proporcionar autenticación basada en claves.

5.9 Canonicalización

Una canonicalización de XML es un método para serializar XML de forma coherente en un flujo de octetos, según sea necesario antes de cifrar XML.

5.9.1 Canonicalización inclusiva

Identificadores:
http://www.w3.org/TR/2001/REC-xml-c14n-20010315 (OPCIONAL)
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments (OPCIONAL)

XML canónico [Canon] es un método de serialización de XML que incluye el espacio de nombres dentro del alcance y el contexto del atributo del espacio de nombres xml de los antepasados del XML que se está serializando.

Si XML se va a cifrar y luego descifrar en un entorno diferente y se desea preservar los enlaces de prefijo del espacio de nombres y el valor de los atributos en el espacio de nombres "xml" de su entorno original, entonces se debe utilizar la versión canónica XML con comentarios del XML. Sea la serialización que esté cifrada.

5.9.2 Canonicalización exclusiva

Identificadores:

http://www.w3.org/2001/10/xml-exc-c14n# (OPCIONAL)
http://www.w3.org/2001/10/xml-exc-c14n#WithComments (OPCIONAL)

Canonicalización XML exclusiva [Exclusivo] serializa XML de tal manera que incluye en la medida mínima práctica el enlace de prefijo de espacio de nombres y el contexto de atributo de espacio de nombres xml heredado de elementos ancestros.

Es el método recomendado donde se puede cambiar el contexto externo de un fragmento que fue firmado y luego cifrado. De lo contrario, la validación de la firma sobre el fragmento puede fallar porque la canonicalización mediante validación de firma puede incluir espacios de nombres innecesarios en el fragmento.

6 consideraciones de seguridad

6.1 Relación con las firmas digitales XML

La aplicación de cifrado y firmas digitales en partes de un documento XML puede dificultar el descifrado y la verificación de la firma posteriores. En particular, al verificar una firma se debe saber si la firma se calculó sobre la forma de elementos cifrados o no cifrados.

Un problema aparte, pero importante, es la introducción de vulnerabilidades criptográficas al combinar firmas digitales y cifrado sobre un elemento XML común. Hal Finney ha sugerido que cifrar datos firmados digitalmente, dejando la firma digital en claro, puede permitir ataques de adivinación de texto sin formato. Esta vulnerabilidad se puede mitigar mediante el uso de hashes seguros y nonces en el texto que se procesa.

De acuerdo con el documento de requisitos [EncReq], la interacción entre cifrado y firma es un problema de aplicación y está fuera del alcance de la especificación. Sin embargo, hacemos las siguientes recomendaciones:

1. Cuando los datos están cifrados, cualquier resumen o firma sobre esos datos debe cifrarse. Esto satisface el primer problema en el sentido de que sólo se pueden validar aquellas firmas que se pueden ver. También aborda la posibilidad de una vulnerabilidad de adivinación de texto sin formato, aunque puede que no sea posible identificar (o incluso conocer) todas las firmas de un determinado dato.
2. Emplee la transformación de firma "decrypt-except" [XML-DSIG-Decrypt] . Funciona de la siguiente manera: durante el procesamiento de transformación de firma, si encuentra una transformación de descifrado, descifre todo el contenido cifrado del documento excepto aquellos exceptuados por un conjunto enumerado de referencias.

Además, si bien las siguientes advertencias se refieren a inferencias incorrectas por parte del usuario sobre la autenticidad de la información cifrada, las aplicaciones deben disuadir la mala interpretación del usuario comunicando claramente qué información tiene integridad o está autenticada, es confidencial o no repudiable cuando se realizan múltiples procesos (por ejemplo, firma). y cifrado) y se utilizan algoritmos (por ejemplo, simétricos y asimétricos):

1. Cuando un sobre cifrado contiene una firma, la firma no necesariamente protege la autenticidad o integridad del texto cifrado [Davis].
2. Si bien la firma protege el texto sin formato, solo cubre lo que está firmado, los destinatarios de los mensajes cifrados no deben inferir la integridad o autenticidad de otra información sin firmar (por ejemplo, encabezados) dentro del sobre cifrado; consulte [XML-DSIG , 8.1.1 Solo lo que está firmado es seguro].

6.2 Información revelada

Cuando una clave simétrica se comparte entre varios destinatarios, esa clave simétrica *solo* debe usarse para los datos destinados a *todos* los destinatarios; Incluso si un destinatario no es dirigido a información destinada (exclusivamente) a otro en la misma clave simétrica, la información podría ser descubierta y descifrada.

Además, los diseñadores de aplicaciones deben tener cuidado de no revelar ninguna información en los parámetros o identificadores de algoritmos (por ejemplo, información en un URI) que debilite el cifrado.

6.3 Nonce y IV (Valor o Vector de Inicialización)

Una característica indeseable de muchos algoritmos de cifrado y/o sus modos es que el mismo texto sin formato, cuando se cifra con la misma clave, tiene el mismo texto cifrado resultante. Si bien esto no es

sorprendente, invita a varios ataques que se mitigan al incluir datos arbitrarios y no repetidos (bajo una clave determinada) con el texto sin formato antes del cifrado. En los modos de encadenamiento de cifrado, estos datos son los primeros en cifrarse y, en consecuencia, se denominan IV (valor de inicialización o vector).

Los diferentes algoritmos y modos tienen requisitos adicionales sobre las características de esta información (por ejemplo, aleatoriedad y secreto) que afectan las características (por ejemplo, confidencialidad e integridad) y su resistencia a los ataques.

Dado que los datos XML son redundantes (por ejemplo, codificaciones Unicode y etiquetas repetidas) y que los atacantes pueden conocer la estructura de los datos (por ejemplo, DTD y esquemas), los algoritmos de cifrado deben implementarse y utilizarse cuidadosamente en este sentido.

Para el modo Cipher Block Chaining (CBC) utilizado por esta especificación, el IV no debe reutilizarse para ninguna clave y debe ser aleatorio, pero no es necesario que sea secreto. Además, en este modo, un adversario que modifique el IV puede realizar un cambio conocido en el texto sin formato después del descifrado. Este ataque se puede evitar asegurando la integridad de los datos de texto sin formato, por ejemplo firmándolos.

6.4 Denegación de Servicio

Esta especificación permite el procesamiento recursivo. Por ejemplo, es posible el siguiente escenario: `EncryptedKey` **A** requiere que `EncryptedKey` **B** sea descifrado, lo que a su vez requiere `EncryptedKey` **A** ! O bien, un atacante podría enviar un mensaje `EncryptedData`de descifrado que haga referencia a recursos de red que son muy grandes o que se redirigen continuamente. En consecuencia, las implementaciones deberían poder restringir la recursividad arbitraria y la cantidad total de recursos de procesamiento y red que una solicitud puede consumir.

6.5 Contenido inseguro

El cifrado XML se puede utilizar para ocultar, mediante cifrado, contenido que las aplicaciones (por ejemplo, cortafuegos, detectores de virus, etc.) consideran inseguro (por ejemplo, código ejecutable, virus, etc.). En consecuencia, dichas aplicaciones deben considerar que el contenido cifrado es tan inseguro como el contenido más inseguro transportado en el contexto de su aplicación. En consecuencia, dichas aplicaciones pueden optar por (1) no permitir dicho contenido, (2) exigir acceso al formulario descifrado para su inspección o (3) garantizar que el contenido arbitrario pueda procesarse de forma segura al recibir las aplicaciones.

7 Conformidad

Una implementación cumple con esta especificación si genera con éxito la sintaxis de acuerdo con las definiciones del esquema y satisface todos los requisitos MUST/REQUIRED/SHALL, incluido el soporte y el procesamiento de algoritmos . Los requisitos de procesamiento se especifican sobre las funciones de ***descifrador* , *cifrador* y** su ***aplicación*** de llamada .

8 Tipo de medio de cifrado XML

8.1 Introducción

Sintaxis y procesamiento de cifrado XML [XML-Encryption] especifica un proceso para cifrar datos y representar el resultado en XML. Los datos pueden ser datos arbitrarios (incluido un documento XML), un elemento XML o contenido de un elemento XML. El resultado del cifrado de datos es un elemento de cifrado XML que contiene o hace referencia a los datos cifrados.

El `application/xenc+xml`tipo de medio permite que las aplicaciones de cifrado XML identifiquen documentos cifrados. Además, permite que las aplicaciones que conocen este tipo de medio (incluso si no son implementaciones de cifrado XML) tengan en cuenta que el tipo de medio del objeto descifrado (original) puede ser un tipo distinto de XML.

8.2 aplicación/xenc+xml Registro

Este es un registro de tipo de medio tal como se define en Extensiones multipropósito de correo de Internet (MIME), parte cuatro: Procedimientos de registro [MIME-REG]

Nombre del tipo de medio MIME: aplicación

Nombre del subtipo MIME: xenc+xml

Parámetros requeridos: ninguno

Parámetros opcionales: juego de caracteres

Los valores permitidos y recomendados y la interpretación del parámetro charset son idénticos a los proporcionados para 'application/xml' en la sección 3.2 de RFC 3023 [XML-MT].

Consideraciones de codificación:

Las consideraciones de codificación son idénticas a las dadas para 'aplicación/xml' en la sección 3.2 de RFC 3023 [XML-MT].

Consideraciones de Seguridad:

Consulte la sección Consideraciones de seguridad [Cifrado XML] .

Consideraciones de interoperabilidad: ninguna

Especificación publicada: [Cifrado XML]

Aplicaciones que utilizan este tipo de medio:

XML Encryption es neutral en cuanto a dispositivos, plataformas y proveedores y es compatible con una variedad de aplicaciones web.

Información adicional:

Número(s) mágico(s): ninguno

Aunque no se puede contar con secuencias de bytes para identificar consistentemente los documentos XML Encryption, serán documentos XML en los que el elemento raíz ' `QNames` `LocalPart`es `'EncryptedData'`o ' `EncryptedKey`' con un nombre de espacio de nombres asociado de ' http://www.w3.org/2001/ 04/xmlenc# '. El `application/xenc+xml`nombre del tipo DEBE usarse solo para objetos de datos en los que el elemento raíz sea del espacio de nombres de cifrado XML. Los documentos XML que contienen estos tipos de elementos en lugares distintos del elemento raíz se pueden describir utilizando funciones como [esquema XML].

Extensiones de archivo: .xml

Código(s) de tipo de archivo de Macintosh: "TEXTO"

Persona y dirección de correo electrónico a contactar para obtener más información:

José Reagle <reagle@w3.org>

Grupo de trabajo XENC <xml-encryption@w3.org>

Uso previsto: COMÚN

Autor/Cambiar controlador:

La especificación de cifrado XML es un producto del trabajo del World Wide Web Consortium (W3C), que tiene control de cambios sobre la especificación.

9 esquema y ejemplos válidos

Esquema
esquema-xenc.xsd
Ejemplo
enc-example.xml (no es criptográficamente válido pero ejercita gran parte del esquema)

10 referencias

TRIPLES
ANSI X9.52: Modos de operación del algoritmo de cifrado de datos triple. 1998.
AES
NIST FIPS 197: Estándar de cifrado avanzado (AES) . Noviembre de 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
ENVOLTURA AES
RFC3394: Algoritmo de ajuste de claves del estándar de cifrado avanzado (AES) . J. Schaad y R. Housley. Informativo, septiembre de 2002.

Algoritmos CMS
RFC3370: Algoritmos de sintaxis de mensajes criptográficos (CMS) . R. Housley. Informativo, febrero de 2002.
http://www.ietf.org/rfc/rfc3370.txt
Envoltura CMS
RFC3217: Encapsulación de claves Triple-DES y RC2 . R. Housley. Informativo, diciembre de 2001.
http://www.ietf.org/rfc/rfc3217.txt
davis
Firma y cifrado defectuosos en S/MIME, PKCS#7, MOSS, PEM, PGP y XML. D. Davis. Conferencia Técnica Anual de USENIX. 2001.
http://www.usenix.org/publications/library/proceedings/usenix01/davis.html
DES
NIST FIPS 46-3: Estándar de cifrado de datos (DES). Octubre de 1999.
http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
EncReq
Requisitos de cifrado XML . J. Reagle. Nota del W3C, marzo de 2002.
http://www.w3.org/TR/2002/NOTE-xml-encryption-req-20020304
ESDH
RFC 2631: Método de acuerdo de claves Diffie-Hellman. E. Rescorla. Seguimiento de estándares, 1999.
http://www.ietf.org/rfc/rfc2631.txt
http://www.w3.org/TR/2002/CR-xml-exc-c14n-20020212
Glosario
RFC 2828: Glosario de seguridad de Internet . R Shirey. Informativo, mayo de 2000.
http://www.ietf.org/rfc/rfc2828.txt
HMAC
RFC 2104: HMAC: hash con clave para autenticación de mensajes . H. Krawczyk, M. Bellare y R. Canetti. Informativo, febrero de 1997.
http://www.ietf.org/rfc/rfc2104.txt
HTTP
RFC 2616: Protocolo de transferencia de hipertexto - HTTP/1.1. J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach y T. Berners-Lee. Standards Track, junio de 1999.
http://www.ietf.org/rfc/rfc2616.txt
PALABRAS CLAVE
RFC 2119: Palabras clave para uso en RFC para indicar niveles de requisitos. S. Bradner. Mejores prácticas actuales, marzo de 1997.
http://www.ietf.org/rfc/rfc2119.txt
MD5
RFC 1321: Algoritmo de resumen de mensajes MD5. R. Rivest. Informativo, abril de 1992.
http://www.ietf.org/rfc/rfc1321.txt
MÍMICA
RFC 2045: Extensiones multipropósito de correo de Internet (MIME), primera parte: Formato de los cuerpos de los mensajes de Internet . N. Freed y N. Borenstein. Standards Track, noviembre de 1996.
http://www.ietf.org/rfc/rfc2045.txt
MIME-REG
RFC 2048: Extensiones multipropósito de correo de Internet (MIME), cuarta parte: Procedimientos de registro . N. Freed, J. Klensin y J. Postel. Mejores prácticas actuales, noviembre de 1996.
http://www.ietf.org/rfc/rfc2048.txt
NFC
TR15, formularios de normalización Unicode . M. Davis y M. Dürst. Revisión 18: noviembre de 1999.
http://www.unicode.org/unicode/reports/tr15/tr15-18.html .
Corrección NFC
"Corrigéndum n.º 2: Yod con normalización de Hiriq ".
http://www.unicode.org/versions/corrigendum2.html .
prop1
"Propuesta de hombre de paja de cifrado XML ". E. Simon y B. LaMacchia. Agosto de 2000.
http://lists.w3.org/Archives/Public/xml-encryption/2000Aug/0001.html
prop2
Otra propuesta de Cifrado XML . T. Imamura. Agosto de 2000.
http://lists.w3.org/Archives/Public/xml-encryption/2000Aug/0005.html
prop3
Sintaxis y procesamiento de cifrado XML . B. Dillaway, B. Fox, T. Imamura, B. LaMacchia, H. Maruyama, J. Schaad y E. Simon. Diciembre de 2000.
http://lists.w3.org/Archives/Public/xml-encryption/2000Dec/att-0024/01-XMLEncryption_v01.html
PKCS1
RFC 2437: PKCS #1: Especificaciones de criptografía RSA Versión 2.0. B. Kaliski y J. Staddon. Informativo, octubre de 1998.
http://www.ietf.org/rfc/rfc2437.txt
ALEATORIO

RFC 1750: Recomendaciones de aleatoriedad para la seguridad . D. Eastlake, S. Crocker y J. Schiller. Informativo, diciembre de 1994.
http://www.ietf.org/rfc/rfc1750.txt
RIPEMD-160
CryptoBytes, Volumen 3, Número 2. La función hash criptográfica RIPEMD-160 . Laboratorios RSA. Otoño de 1997.
ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto3n2.pdf
http://www.esat.kuleuven.ac.be/~cosicart/pdf/AB-9601/AB-9601.pdf
sha
Estándar de hash seguro . NIST FIPS 180-1. (RFC 3174). Abril de 1995.
http://www.itl.nist.gov/fipspubs/fip180-1.htm
Estándar de hash seguro. Borrador NIST FIPS 180-2 . 2001. (Ampliado para incluir SHA-384, SHA-256 y SHA-512)
http://csrc.nist.gov/encryption/shs/dfips-180-2.pdf
A Bin
R. Tobin. Conjunto de información para entidades externas , lista de correo XML Core, 2000 [solo miembro del W3C].
http://lists.w3.org/Archives/Member/w3c-xml-core-wg/2000OctDec/0054
UTF-16
RFC 2781: UTF-16, una codificación de ISO 10646. P. Hoffman y F. Yergeau. Informativo, febrero de 2000.
http://www.ietf.org/rfc/rfc2781.txt
UTF-8
RFC 2279: UTF-8, un formato de transformación de ISO 10646F. F. Yergeau. Standards Track, enero de 1998.
http://www.ietf.org/rfc/rfc2279.txt
URI
RFC 2396: Identificadores uniformes de recursos (URI): sintaxis genérica . T. Berners-Lee, R. Fielding y L. Masinter. Standards Track, agosto de 1998.
http://www.ietf.org/rfc/rfc2396.txt
http://www.ietf.org/rfc/rfc1738.txt
http://www.ietf.org/rfc/rfc2141.txt
RFC 2611: Mecanismos de definición de espacios de nombres URN. Mejores prácticas actuales. Daigle, D. van Gulik, R. Iannella, P. Falstrom. Junio de 1999.
http://www.ietf.org/rfc/rfc2611.txt
X509v3
Recomendación UIT-T X.509 versión 3 (1997). "Tecnología de la información - Interconexión de sistemas abiertos - El marco de autenticación de directorios" ISO/IEC 9594-8:1997.
XML
Lenguaje de marcado extensible (XML) 1.0 (segunda edición) . T. Bray, J. Paoli, CM Sperberg-McQueen y E. Maler. Recomendación del W3C, octubre de 2000.
Base XML
Base XML . J. Marsh. Recomendación del W3C, junio de 2001.
http://www.w3.org/TR/2001/REC-xmlbase-20010627/
XML-C14N
XML canónico. J. Boyer. Recomendación del W3C, marzo de 2001.
http://www.w3.org/TR/2001/REC-xml-c14n-20010315
http://www.ietf.org/rfc/rfc3076.txt
XML-exc-C14N
Canonicalización XML exclusiva . J. Boyer, D. Eastlake y J. Reagle. Recomendación del W3C, julio de 2002.
http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/
XML-DSIG
Sintaxis y procesamiento de firmas XML . D. Eastlake, J. Reagle y D. Solo. Recomendación del W3C, febrero de 2002.
http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/
XML-DSIG-Descifrar
Transformación de descifrado para firma XML . M. Hughes, T. Imamura y H. Maruyama. Recomendación del W3C, diciembre de 2002.
http://www.w3.org/TR/2002/REC-xmlenc-decrypt-20021210
Cifrado XML
Sintaxis y procesamiento de cifrado XML . D. Eastlake y J. Reagle. Recomendación candidata del W3C, diciembre de 2002.
http://www.w3.org/TR/2002/CR-xmlenc-core-20020802/
Conjunto de información XML
Conjunto de información XML . J. Cowan y R. Tobin. Recomendación del W3C, octubre de 2001
http://www.w3.org/TR/2001/REC-xml-infoset-20011024/
XML-MT
RFC 3023: tipos de medios XML. M. Murata, S. St. Laurent y D. Kohn. Informativo, enero de 2001.
http://www.ietf.org/rfc/rfc2376.txt
XML-NS

Espacios de nombres en XML . T. Bray, D. Hollander y A. Layman. Recomendación del W3C, enero de 1999.
http://www.w3.org/TR/1999/REC-xml-names-19990114

esquema XML

Esquema XML Parte 1: Estructuras D. Beech, M. Maloney y N. Mendelsohn. Recomendación del W3C, mayo de 2001.
http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
Esquema XML, parte 2: tipos de datos . P. Biron y A. Malhotra. Recomendación del W3C, mayo de 2001.
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

XPath

Lenguaje de ruta XML (XPath) versión 1.0 . J. Clark y S. DeRose. Recomendación del W3C, octubre de 1999.
http://www.w3.org/TR/1999/REC-xpath-19991116